MASTER EN SCIENCES MATHÉMATIQUES

Médecine nucléaire : optimisation dans le cadre du recalage d'images médicales

DELAHAUT, Céline

Award date:
2014

UNIVERSITE DE NAMUR

Faculté des Sciences

MEDECINE NUCLEAIRE : OPTIMISATION DANS LE CADRE DU RECALAGE D'IMAGES MEDICALES

Promoteur : Annick SARTENAER
Co-promoteur : Hubert MEURISSE

Mémoire présenté pour l'obtention

du grade académique de master en Sciences mathématiques à finalité spécialisée

Céline DELAHAUT

Juin 2014

Résumé

Dans le cadre de l'imagerie médicale, il est parfois nécessaire de superposer des images obtenues par différentes méthodes d'acquisition et donc de modalités différentes (fonctionnelles ou anatomiques) pour enrichir l'information. Pour ce faire, les médecins utilisent des algorithmes de *recalage* qui réalignent les différentes images synchronisées pour ensuite les sommer. Dans ces algorithmes, l'optimisation joue un rôle important car elle permet de minimiser la distance entre les deux images qui est caractérisée par un critère de ressemblance. Pour sa part, le service de Médecine Nucléaire du CHU de Mont-Godinne utilise la librairie ITK qui fournit de nombreux outils de segmentation et de recalage. Parmi toutes les méthodes que cette librairie propose, le CHU utilise, entre autres, l'algorithme de recalage des démons. Cet algorithme peut être considéré comme un processus d'optimisation. Le but de ce mémoire est, dans un premier temps, de se familiariser avec l'imagerie médicale, la librairie ITK et l'algorithme des démons. Dans un second temps, il consiste à étudier l'implémentation de cet algorithme et, plus particulièrement, les méthodes d'optimisation qui y sont appliquées.

Mots-clés : *Imagerie médicale* ; *algorithme de recalage* ; *librairie ITK* ; *méthode des démons* ; *optimisation.*

Abstract

In the context of medical imaging, it is sometimes necessary to superimpose images obtained by different methods of acquisition and therefore different modalities (functional or anatomical) to enrich the information. To do this, doctors use matching algorithms that realign the different registered images and, subsequently, sum them together. In these algorithms, optimization plays an important role because it allows to minimize the distance between two images which is characterized by a criterion of similarity. On its side, the Nuclear Medicine department of the university hospital of Mont-Godinne (CHU) uses the ITK library that provides many tools for segmentation and registration. Among the methods that the library offers, the CHU uses, among other things, the registration demons algorithm. This algorithm can be seen as an optimization procedure. The purpose of this thesis is first to deal with medical imaging, the ITK library and the demons algorithm. It further studies the implementation of this algorithm and, more specifically, the optimization techniques used therein.

Keywords : *Medical imaging* ; *matching algorithm* ; *ITK library* ; *method of the demons* ; *optimization.*

Remerciements

En préambule de ce mémoire, je souhaite présenter mes remerciements les plus sincères aux personnes qui m'ont apporté leur aide et qui ont contribué à l'élaboration de ce mémoire ainsi qu'à la réussite de cette formidable année universitaire.

Je tiens à remercier tout naturellement Madame Annick Sartenaer, professeur au Département de Mathématique de l'Université de Namur et promotrice de ce mémoire, qui s'est toujours montrée à l'écoute et très disponible tout au long de la réalisation de ce mémoire malgré ses charges académiques. Sans l'inspiration, l'aide et le temps qu'elle a bien voulu me consacrer, ce mémoire n'aurait jamais vu le jour.

Je remercie également de tout coeur Monsieur Hubert Meurisse, co-promoteur de ce mémoire, Radio-physicien et ingénieur biomédical au Service Nucléaire du Centre Hospitalier Universitaire de Mont-Godinne, ainsi que Monsieur Damien De Nizza, membre de la Faculté d'Informatique de l'Université de Namur. Leur disponibilité et la rapidité avec laquelle ils répondaient aux emails étaient sans limite. Leurs conseils étaient toujours très judicieux.

En tant que débutante dans le développement de programmes d'imagerie médicale, il a été nécessaire pour moi de fouiller dans la documentation mise sur Internet concernant ce sujet. J'aimerais donc remercier la communauté *ITK* pour leur aide dans la compréhension et l'utilisation de leur librairie.

Au terme de mes études, j'adresse mes remerciements au corps professoral (assistants et professeurs) et administratif du Département de Mathématique de l'Université de Namur. Ils offrent un enseignement riche et de qualité et déploient également de grands efforts pour assurer à leurs étudiants une formation actualisée.

D'un point de vue plus personnel, un grand merci à tous mes proches et amis, qui m'ont soutenue et encouragée durant l'ensemble de mes études. En particulier, je tiens à remercier mes parents qui m'ont donné la possibilité de suivre ces études, et ce dans des conditions idéales.

Merci enfin à Grégory, mon compagnon, qui m'a soutenue et supportée tout au long de la rédaction de ce mémoire, et qui m'a apporté sa connaissance de la langue française pour la relecture de ce mémoire.

Merci à toutes et à tous

Céline Delahaut

Table des matières

Chapitre 1

Introduction

Nous commençons par présenter le contexte du mémoire, c'est-à-dire les différentes étapes de la collaboration avec le service de Médecine Nucléaire du CHU de Mont-Godinne. Nous exposons ensuite le contenu de ce mémoire avec les grands concepts utilisés et la structure des différents chapitres.

1.1 Contexte du mémoire

Suite aux mémoires de Stéphane Calande [7], de Sébastien Wilfart [61] et d'Yvan Michiels [27], réalisés en collaboration avec le CHU de Mont-Godinne, un nouveau sujet a été proposé dans le cadre d'une collaboration entre naXys ("Namur Center For Complex Systems"), NARILIS ("Namur Research Institute for Life Sciences") et le CHU de Mont-Godinne. Ce projet de mémoire avait pour titre "Le traitement numérique d'images 3D en imagerie médicale" et proposait un encadrement multidisciplinaire.

Comme premier contact avec le CHU, une réunion s'est organisée avec la présence d'Annick Sartenaer (promotrice de ce mémoire), de Timoteo Carletti (responsable de naXys), de Damien De Nizza (membre de la Faculté d'Informatique), de Stéphane Lucas (membre de NARILIS), d'Hubert Meurisse (co-promoteur de ce mémoire et membre du service de Médecine Nucléaire du CHU), de deux autres médecins et moi-même. Lors de cette réunion, le contexte de l'imagerie médicale nous a été présenté, ainsi que l'un ou l'autre projet pour lesquels nos compétences de mathématicien leurs semblaient importantes. Le premier projet présenté avait pour objet le cartilage du genou et le deuxième projet concernait la moelle épinière.

Par la suite, Hubert Meurisse nous a invitées, ma promotrice et moi-même, au service de Médecine Nucléaire pour une bonne entrée en matière. Lors de cette rencontre, nous avons assisté à une présentation d'une médecin assistante en Médecine Nucléaire sur l'intérêt de comparer les images de scintigraphie pulmonaire avec celles du scanner. De plus, dans le cadre de son mémoire, Hasan Cavus, étudiant en physique médicale à l'Université de Liège, nous a ensuite présenté le recalage d'images dans les embolies pulmonaires. Grâce à ces présentations, nous nous sommes familiarisées avec les concepts d'imagerie médicale et de recalage d'images, et mon sujet de mémoire s'est précisé, à savoir *"L'optimisation dans le cadre du recalage d'images médicales"*.

La machine étant mise en route, nous pouvions avancer sans l'aide continue du CHU. Pendant toute la réalisation de ce mémoire, nous avons cependant rencontré, à plusieurs reprises, Hubert Meurisse, Damien De Nizza et Hasan Cavus pour s'assurer de comprendre tous les concepts nécessaires et pour décider des différentes directions à prendre. Dans la section suivante, nous introduisons le contenu de ce mémoire avec les notions d'imagerie médicale et de recalage d'images mentionnées ci-dessus.

1.2 Contenu du mémoire

Au fil des années, la médecine n'a cessé de se développer et a rencontré le besoin grandissant de pouvoir voir à l'intérieur du corps humain sans avoir à utiliser des techniques dites invasives, comme l'incision. Pour ce faire, il existe à l'heure actuelle plusieurs méthodes, chacune ayant son domaine de prédilection. Parmi les plus répandues, nous pouvons citer entre autres les rayons X, l'imagerie par résonance magnétique et la tomographie par émission de positons. Ces diverses techniques constituent aujourd'hui un domaine très important de la médecine moderne qui est, l'*imagerie médicale*. De nos jours, l'imagerie médicale est présente aussi bien dans le diagnostic, la planification de traitement, le suivi thérapeutique que dans la simulation opératoire.

Dans le cadre de l'imagerie médicale, il existe de nombreux processus appartenant au traitement d'images, dont le recalage d'images. Le *recalage d'images* est le processus d'alignement des images de sorte que les fonctions correspondantes peuvent être facilement liées afin de comparer ou combiner les informations respectives des images. Dans le domaine médical, les professionnels utilisent le recalage pour aider à de nombreuses tâches essentielles telles que le diagnostic, la planification de radiothérapie ou de chirurgie, le positionnement du patient avant la radiothérapie ou la chirurgie, la vérification de l'administration de dose, la surveillance de la progression ou la réponse au traitement des maladies, et la création de modèles de référence.

Le recalage aligne des images en appliquant des transformations à l'une des images pour qu'elle corresponde à l'autre. Il existe deux types de recalage, il peut être rigide ou déformable. Les méthodes de recalage rigide ne permettent que des transformations rigides et les méthodes de recalage déformable appliquent des transformations élastiques pour corriger les déformations que les méthodes rigides ne peuvent pas corriger.

Les applications de l'imagerie médicale utilisent différentes méthodes de recalage déformable. Parmi ces méthodes, les médecins utilisent fréquemment la technique de recalage déformable proposée par J.-P. Thirion [42] appelée *méthode des démons*. Cette méthode est la plus utilisée dû à son efficacité et à sa simplicité. La technique des démons prend deux images en entrée et produit le champ de déplacement qui indique les transformations qui doivent être appliquées aux pixels de l'une des images de manière à pouvoir être alignée avec l'autre.

Afin de trouver la meilleure transformation qui aligne le plus exactement possible les deux images, il faut minimiser la distance entre ces deux images. Cette distance est caractérisée par un critère de similarité. L'*optimisation* joue donc un rôle essentiel dans le processus de recalage d'images. Dans le cadre de ce mémoire, nous nous intéressons tout d'abord à l'optimisation dans le recalage d'images de manière générale, ensuite dans le recalage déformable avec la méthode

des démons.

Le présent chapitre de ce mémoire constitue l'introduction du contexte et du contenu. Dans le deuxième chapitre, nous exposons le cadre de travail dans lequel nous nous positionnons, à savoir l'imagerie médicale et plus particulièrement, les techniques d'acquisition. Le chapitre suivant présente le recalage d'images qui appartient à la famille des techniques du traitement d'images. Nous y expliquons également les différents types de recalage qui existent et nous présentons plusieurs exemples illustratifs afin de faciliter la compréhension du sujet. Ensuite, le chapitre intitulé *Optimisation* consiste en une présentation théorique de l'optimisation avec et sans contraintes. Ce chapitre introduit également la théorie concernant les méthodes d'optimisation utilisées dans la suite. Dans le cinquième chapitre, le lecteur pourra prendre connaissance des moyens informatiques et techniques utilisés dans le cadre de ce mémoire. De plus, il comprendra les méthodes d'optimisations les plus utilisées dans le recalage d'images médicales. Le sixième chapitre se concentrera sur la méthode de recalage utilisée par le CHU de Mont-Godinne, la méthode des démons. Nous y présentons les grands principes de cette méthode et les techniques utilisées pour la développer. Enfin, nous ferons le lien entre la théorie et la pratique. Le septième chapitre constitue la conclusion et les perspectives possibles suite à ce mémoire.

Chapitre 2

Introduction à l'imagerie médicale

Ce chapitre a pour objectif de dresser le cadre dans lequel s'intègre ce mémoire : l'*imagerie médicale*. L'imagerie médicale est un domaine complexe, en pleine effervescence à l'heure actuelle et qui utilise de nombreux concepts. Ce chapitre introduit les notions nécessaires pour comprendre le domaine d'intervention.

2.1 L'imagerie médicale

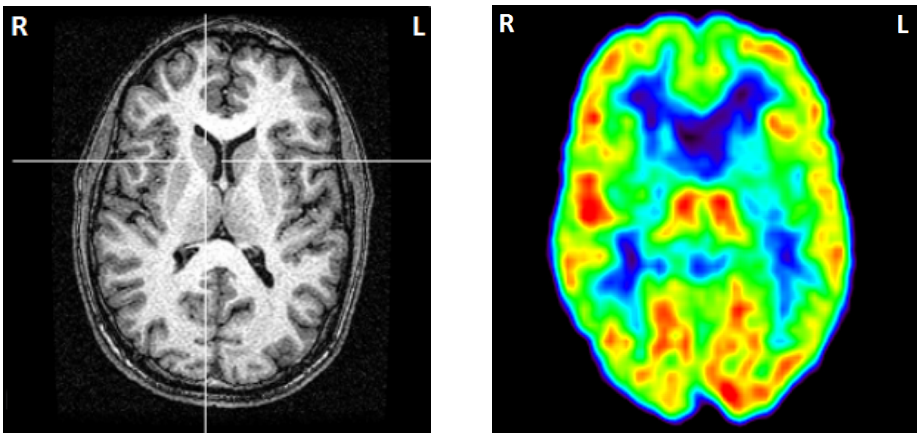
Selon Wikipédia [53], l'imagerie médicale est l'ensemble des techniques permettant aux médecins d'examiner l'intérieur du patient sans avoir recours à une intervention chirurgicale. L'imagerie médicale regroupe donc les *méthodes d'acquisition* et *de restitution* d'images du corps humain à partir de différents phénomènes physiques tels que l'absorption de rayons X, la résonance magnétique nucléaire, la réflexion d'ondes ultrasons ou la radioactivité. Ces technologies d'acquisition d'images ont révolutionné la médecine car elles permettent de visualiser indirectement l'*anatomie* (organes, tissus et cellules [49]), la *physiologie* (fonctionnement et organisation mécanique, physique et biochimique des composants [57]) ou le *métabolisme* (ensemble des transformations moléculaires et énergétiques qui se déroulent de manière ininterrompue dans les cellules [55]) du corps humain.

Le but de l'imagerie médicale est de créer une représentation visuelle intelligible d'une information à caractère médical. Autrement dit, l'objectif est de pouvoir représenter, sous un format relativement simple, une grande quantité d'informations issues d'une multitude de mesures acquises selon une modalité bien définie. Dans le cadre de l'imagerie médicale, nous parlons de *modalités* pour désigner les différentes techniques d'acquisition d'images médicales. Dans un sens plus large, le domaine de l'imagerie médicale englobe toutes les techniques permettant de stocker et de manipuler ces informations. Ainsi, il existe une norme pour la gestion informatique des données issues de l'imagerie médicale : la norme DICOM qui sera expliquée par la suite.

2.1.1 Les différentes techniques d'acquisition

Comme nous l'avons mentionné, l'imagerie médicale regroupe de nombreuses techniques d'acquisition d'informations différentes. Selon les techniques utilisées, nous parlerons d'*imagerie structurelle* (ou *morphologique*) ou d'*imagerie fonctionnelle*. Dans le premier cas, les examens d'imagerie médicale permettent d'obtenir des informations sur l'anatomie des organes (leur

taille, leur volume, leur localisation, la forme d'une éventuelle lésion, etc.). Dans le second cas, les examens d'imagerie médicale fournissent des informations sur le *fonctionnement* des organes (leur physiologie, leur métabolisme, etc.). La Figure 2.1 illustre l'anatomie du cerveau grâce à une technique d'imagerie structurelle (Figure 2.1(a)) et le fonctionnement du cerveau grâce à une technique d'imagerie fonctionnelle (Figure 2.1(b)).



(a) Imagerie structurelle (b) Imagerie fonctionelle

FIGURE 2.1 – Images médicales du cerveau [22]

Parmi les méthodes d'imagerie structurelle employées en médecine nucléaire, nous pouvons citer :

- les méthodes basées sur le phénomène d'atténuation des rayons X par les différents milieux biologiques traversés (radiologie conventionnelle, radiologie digitale, tomodensitométrie, CT-scan pour Computed Tomography scanner ou X-Ray qui signifie Rayon X),
- les méthodes basées sur le phénomène de résonance magnétique nucléaire (IRM qui signifie Imagerie par Résonance Magnétique),
- les méthodes échographiques qui utilisent les ultra-sons, et
- les méthodes optiques qui utilisent les rayons lumineux.

Sur la Figure 2.1, l'image de gauche est obtenue par une méthode d'imagerie structurelle qui est le CT-scan.

Les méthodes d'imagerie fonctionnelle quant-à-elles regroupent, entre autres :

- les techniques de médecine nucléaire basées sur l'émission de rayons gamma par des traceurs radioactifs qui, après injection, se concentrent dans les régions d'intense activité métabolique (PET-scan qui signifie Tomographie par Emission de Positons ou TEMP-scan pour Tomographie d'Emission MonoPhotonique),
- les techniques électro-physiologiques qui mesurent les modifications de l'état électrochimique des tissus, ou
- les techniques issues de l'IRM dite fonctionnelle.

Sur la Figure 2.1, l'image de droite est obtenue par une méthode d'imagerie fonctionnelle qui est le PET-scan.

L'ensemble de ces techniques ne donne pas les mêmes informations sur le corps du patient, elles sont complémentaires. Il est donc intéressant de combiner les informations obtenues par différentes méthodes d'acquisition. C'est le domaine de l'*imagerie multimodale*, où les données issues de plusieurs techniques sont mises en commun et *recalées*, c'est-à-dire mises en correspondance au sein d'un même document. La Figure 2.2 illustre la multimodalité structurelle et fonctionnelle pour la compréhension du cerveau humain. On y voit trois images d'imagerie structurelle données par un CT-scan (CT), une IRM et des rayons X (X-Ray), ainsi que trois images d'imagerie fonctionnelle fournies par un PET-scan (PET), TEMP-scan (TEMP) et une IRM fonctionnelle (IRMf).

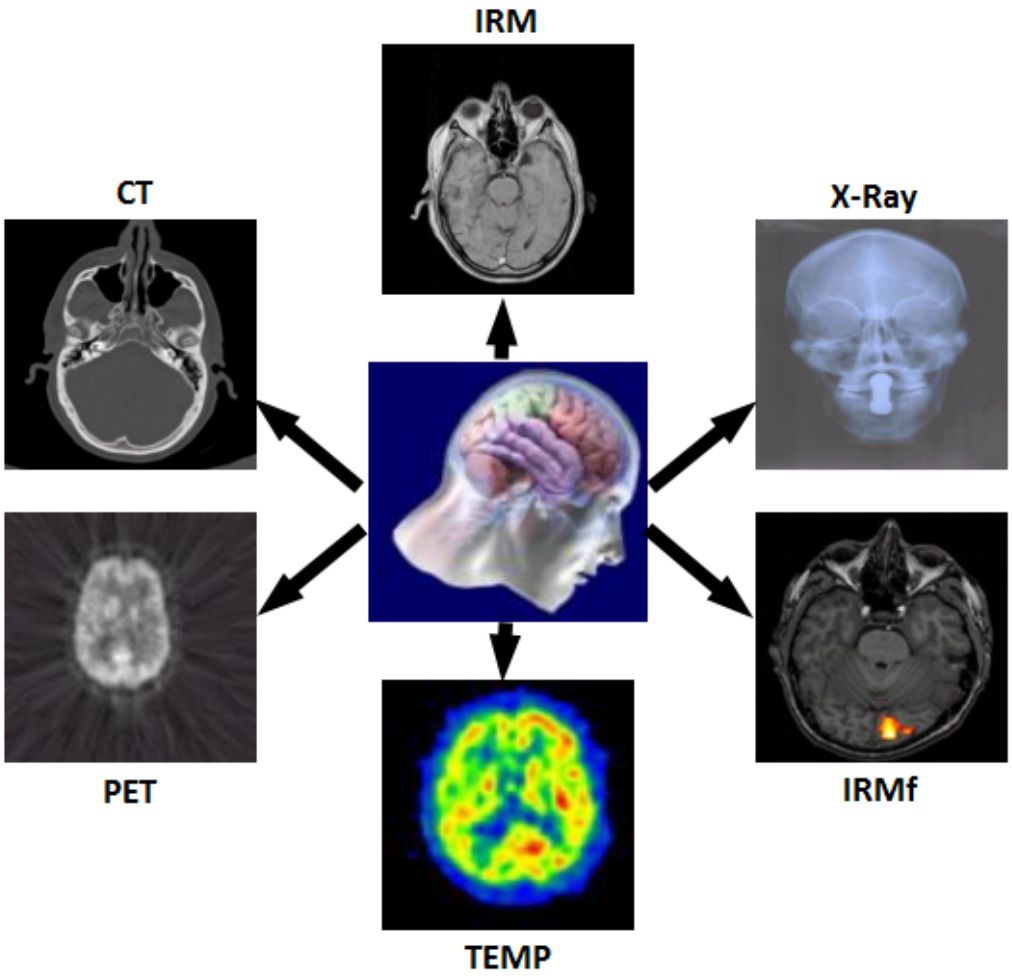


FIGURE 2.2 – Multimodalité de l'imagerie médicale [1]

2.1.2 Système de référence

Un *système de référence* en anatomie est un ensemble de plans et d'axes définis par rapport à la position standard de l'organisme décrit (debout face à l'observateur) [59]. Ce système de référence permet de désigner sous quel angle un volume tridimensionnel est observé. Comme indiqué sur la Figure 2.3, nous distinguons trois axes de référence chez l'être humain :

- le *plan sagittal*, qui sépare la moitié gauche et la moitié droite du corps,
- le *plan transversal* (ou *axial*), qui sépare le corps en une partie supérieure (du côté de la tête) et une partie inférieure (du côté des pieds), et
- le *plan frontal* (ou *coronal*), qui sépare le corps en une partie antérieure ou ventrale et une partie postérieure ou dorsale.

De plus, selon la convention radiologique en imagerie médicale, la partie droite de l'organisme est représentée sur la gauche de l'image et inversément, comme illustré sur la Figure 2.1.

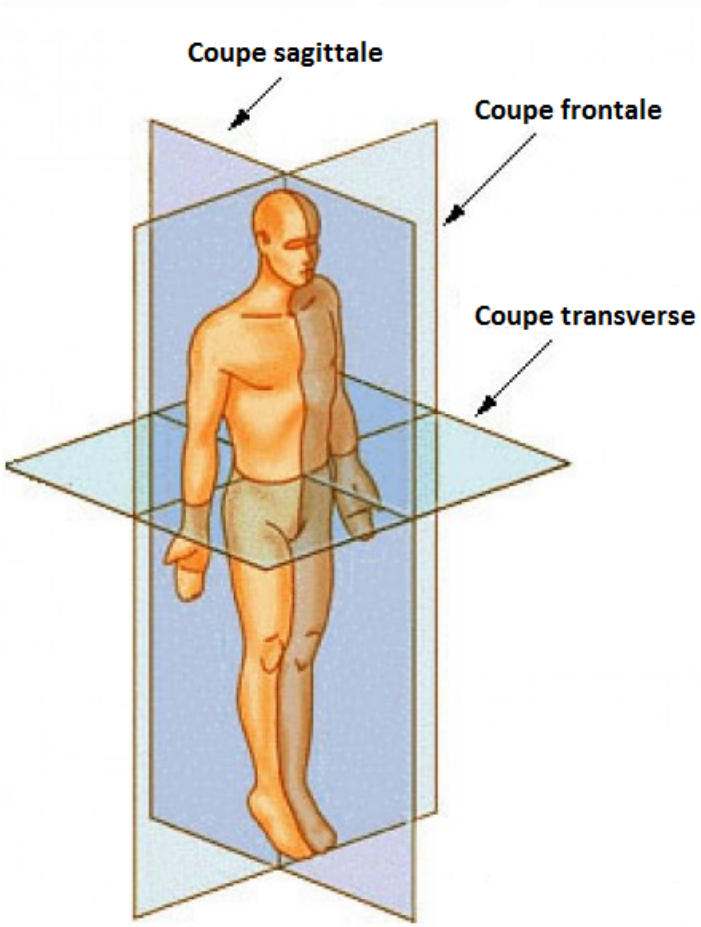


FIGURE 2.3 – Système de référence anatomique [59]

2.1.3 Visualisation

Les processus de l'imagerie médicale (IRM, CT-scan, PET-scan, etc.) permettent d'étudier au travers d'images tridimensionnelles la structure interne et le métabolisme des organes du patient. Ces techniques d'acquisition procèdent par découpage de la structure anatomique à étudier en coupes selon un axe principal du système de référence anatomique. Les images médicales, obtenues par les différentes méthodes d'acquisition, sont donc bidimensionnelles. Dès lors, lorsque nous parlons d'images tridimensionnelles en imagerie médicale, il s'agit d'une

reconstruction effectuée en superposant une séquence de coupes bidimensionnelles. De cette manière, il est possible d'observer un organe soit dans son ensemble sans devoir passer d'une coupe à l'autre grâce à la visualisation d'un volume, soit en détails grâce à la visualisation par coupes.

2.2 Concepts d'imagerie

Pour les concepts d'imagerie, nous nous basons sur la présentation qu'en donnent S. Calande [7] et Y. Michiels [27] dans leur mémoire de fin d'études.

Contrairement à une simple photo de famille, une image médicale doit être précise afin de porter un diagnostic de façon certaine. Une telle image est donc toujours accompagnée d'informations permettant de connaître précisément l'espace physique qu'elle représente. De cette manière, on connaît l'emplacement exact des objets représentés, ainsi que leur taille dans le monde réel.

Dans le cas d'appareils de type CT-scan ou PET-scan, une image représente le volume du corps humain, c'est donc une image en trois dimensions. C'est pourquoi, chaque appareil d'imagerie (CT-scan, PET-scan, IRM, etc.) est associé à un *système d'axes orthonormés*. On peut donc voir le volume qu'une image représente comme un parallélépipède rectangle dont chaque côté est parallèle à un axe. Ce parallélépipède rectangle contient le sujet de l'image. Afin de permettre la correspondance avec le monde réel, une image est caractérisée par :

- *une origine* : L'origine représente le point de départ de la région d'intérêt et est un des sommets du parallélépipède rectangle. L'origine est caractérisée par des coordonnées relatives au système d'axes orthonormés avec le millimètre pour unité ;
- *un spacing* : Comme une image est constituée d'un ensemble de pixels [1], le spacing est la taille d'un pixel en millimètre sur chaque axe. Le spacing peut varier sur chacun des axes. De cette manière, un pixel peut être un parallélépipède et pas un cube. Pour finir, plus le spacing est petit, plus la précision est grande ;
- *un nombre de pixels* sur chacun des axes. Ce nombre de pixels dépend de la taille de la région d'intérêt et du spacing ;
- *une matrice de pixels* : C'est une matrice à trois dimensions qui contient les valeurs d'intensité de chaque pixel de l'image.

2.3 La norme DICOM

Les images acquises par CT-scan ou PET-scan sont encapsulées dans des fichiers qui respectent la *norme DICOM* (Digital Imaging Communication in Medicine, ou imagerie et communication numérique à des fins médicales). Ces fichiers contiennent de nombreuses informations.

1. En trois dimensions, un pixel est appelé voxel. Cependant, le terme pixel est plus familier.

2.3.1 Origines et objectifs

Comme l'affirme S. Seda [39], les évolutions constantes des systèmes d'acquisition d'images ainsi que des systèmes d'archivage et d'informations dans le cadre médical, ont produit dans les années 80 d'importants besoins d'interaction entre les équipements médicaux. Afin d'aider à la manipulation et à la visualisation d'images, les professionnels du médical et les fabricants d'équipements médicaux ont développé le standard DICOM. La norme DICOM a été créée par l'ACR (American College of Radiology) en association avec la NEMA (National Electrical Manufactures Association). Cette norme définit un *format d'images* adapté au contexte médical mais également un *protocole de transmission* des données entre différentes entités.

L'objectif du standard DICOM est donc de faciliter les transferts d'images entre les machines de différents constructeurs. Avant la généralisation de ce format, chaque constructeur de matériel d'imagerie utilisait un format de données différent, entrainant d'importants problèmes de gestion et de maintenance dans les établissements de santé comme des incompatibilités, des coûts supplémentaires ou des pertes d'informations. De plus, le suivi médical de patients souffrant de pathologie lourde, nécessite souvent le transfert d'un établissement de santé à un autre en fonction des moyens et des compétences disponibles. Ce type de transfert a directement bénéficié de l'instauration de la norme. Les images au format DICOM accompagnant les dossiers médicaux sont lisibles sur tout matériel informatique compatible à ce format [52].

2.3.2 Le format d'images

En plus de contenir les pixels qui composent l'image, les fichiers DICOM contiennent de nombreuses informations telles que, par exemple, des informations concernant le patient qui a subi l'examen (nom, prénom, âge, sexe, etc.), des informations sur l'appareil d'acquisition (marque, modèle, etc.), le médecin responsable et la date de l'examen. L'ensemble de ces informations est structuré sous la forme d'une suite de champs. Chaque champ est caractérisé par :

- *Une étiquette (tag).* C'est un code permettant d'identifier le champ. Ce code est constitué de deux parties : le numéro du *groupe* et le numéro de l'*élément*. Une étiquette est, par exemple, [0010-0040], où le numéro du groupe (0010) représente toutes les données concernant le patient et le numéro de l'élément (0040) désigne le sexe du patient. A cette étiquette est lié un libellé qui est l'intitulé de l'information. Dans l'exemple ci-dessus, le libellé est "Patient's Sex". Comme les fichiers DICOM contiennent beaucoup d'informations, il existe de nombreux groupes et éléments. La Table 2.1 indique les groupes d'informations les plus fréquemment utilisés.
- *Une représentation de valeur* qui est une chaîne de deux caractères et qui précise le type de l'information. Par exemple, l'information sur le sexe d'un patient est représenté par CS qui signifie "Code String".
- *Une longueur de la valeur.* C'est la longueur de l'information qui va être lue.
- *Une valeur.* L'information en elle-même.

Les Figures 2.4 et 2.5 illustrent une image IRM du cerveau d'un patient ainsi qu'un aperçu de champs DICOM concernant les informations du patient dans le logiciel DicomWorks [10]. Grâce aux champs DICOM concernant le patient, nous savons que le patient est une femme de 28 ans qui pèse 61 kg.

Groupe	Signification
0008	Identification du centre (date d'examen, type d'examen, fabricant de la machine, hôpital, identification de la machine, etc.)
0010	Informations sur le patient (nom, identification, date de naissance, sexe, etc.)
0018	Informations sur l'acquisition de l'information (épaisseur de coupe, temps d'écho, position du patient, etc.)
0020	Positionnement et informations relatives à l'acquisition (orientation du patient, plan de références, nombre de séries, nombres d'images dans l'acquisition, etc.)
0028	Présentation de l'image (dimensions : largeur et hauteur, niveaux de gris, tables de couleurs, bits alloués, bits stockés, bit le plus significatif, etc.)
4000	Texte (commentaires)
7FE0	Représentation numérique de l'image elle-même (ensemble des pixels de l'image où le premier pixel est celui situé dans le coin supérieur gauche)

TABLE 2.1 – Principaux groupes de champs DICOM [39]

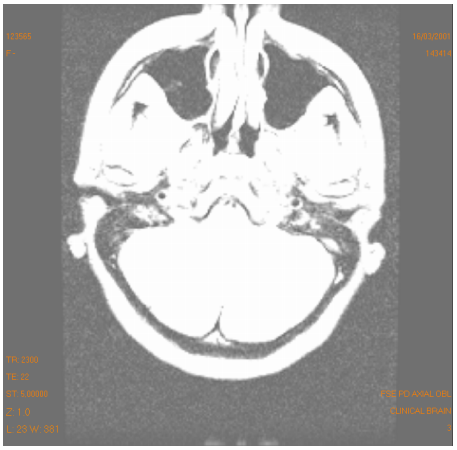


FIGURE 2.4 – Image IRM d'un patient [10]

2.4 Le format MetaImage

Malgré ses nombreux avantages, le format d'images DICOM ne permet pas le traitement de volumes facilement. Chaque fichier représente une coupe du volume considéré et une multitude de fichiers doivent être manipulés pour traiter le volume dans son ensemble. Dans certains cas, il est donc nécessaire de passer du DICOM au *format MetaImage* afin de traiter un volume en entier au lieu de nombreux fichiers représentant chacun une coupe du volume.

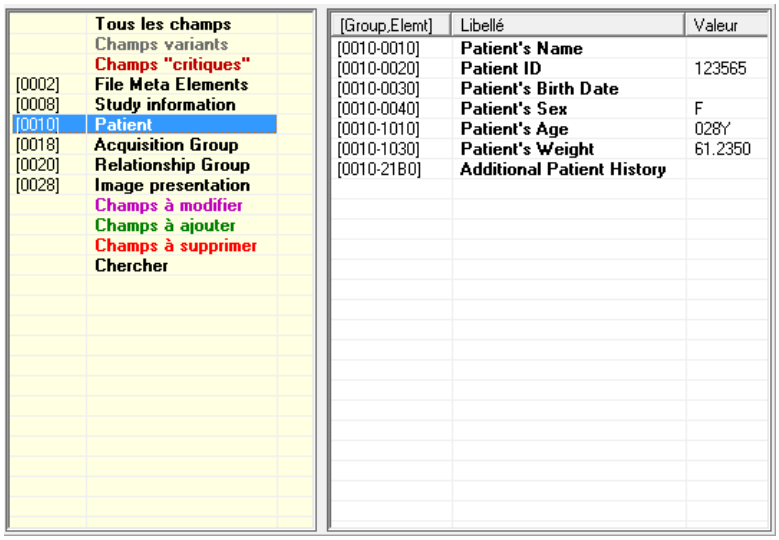


FIGURE 2.5 – Quelques champs DICOM associés à l'IRM de la Figure 2.4 [10]

Selon D. Sarrut [37], une image, enregistrée sous le format MetaImage, se décompose en deux fichiers : *"nomFichier.mhd"* et *"nomFichier.raw"*. Le premier fichier, dont l'extension mhd signifie "Meta Header Data", est un fichier d'*en-tête*. C'est un fichier texte qui contient les informations sur l'image : résolution, origine de la grille, type de codage, etc. La Table 2.2 illustre la composition minimale d'un ficher mhd. En effet, le fichier mhd se compose d'une succession de balises dont certaines sont obligatoires :

- *ObjectType :* le type d'objet, ici l'objet est une "Image" ;
- NDims : le nombre de dimensions de l'image, en général 2 (dans le cas d'une coupe) ou 3 (dans le cas d'un volume) ;
- *DimSize :* la taille de l'image en pixels selon chaque direction (x, y et éventuellement z) ;
- *ElementType :* le type de codage des pixels. Dans l'exemple de la Table 2.2, il s'agit de "USHORT" (entier court non signé), chaque pixel ayant ainsi une valeur comprise entre 0 et 65535 ;
- *ElementDataFile :* une chaîne de caractères qui indique le nom du fichier contenant les données des pixels de l'image. Cette balise doit toujours se trouver à la fin du fichier d'en-tête et le fichier contenant les données des pixels doit se trouver dans le même répertoire que celui d'en-tête.

D'autres balises sont optionnelles mais souvent utilisées et très pratiques. La Table 2.3 illustre un fichier mhd avec les balises optionnelles suivantes :

- *BinaryDataByteOrderMSB (booléen)* : cette balise indique comment les données sont encodées. Cette information est primordiale pour interpréter les données représentant l'image ;
- *Offset :* la position de l'origine de l'image par rapport au système d'axes, avec le millimètre comme unité ;
- *ElementSpacing :* la résolution dans chaque direction. En d'autres mots, c'est la taille, en millimètre, d'un pixel dans chaque direction.

```
ObjectType = Image
NDims = 3
DimSize = 256 256 64
ElementType = MET_USHORT
ElementDataFile = nomFichier.raw
```

TABLE 2.2 – Structure minimale d'un fichier "mhd" [48]

```
ObjectType = Image
NDims = 3
BinaryDataByteOrderMSB = False
Offset = 0 0 0
ElementSpacing = 0.976562 0.976562 2
DimSize = 512 512 141
ElementType = MET_SHORT
ElementDataFile = nomFichier.raw
```

TABLE 2.3 – Structure d'un fichier "mhd" [48]

Le deuxième fichier, d'extension raw, contient les données binaires brutes des pixels. Les valeurs des pixels sont inscrites dans le fichier à la suite l'une de l'autre et sont codées selon les informations spécifiées dans le fichier mhd.

Il existe une variante au format MetaImage qui consiste à coder toute l'image (caractéristiques et données brutes) dans un seul fichier dont l'extension est mha. Ce fichier contient donc le fichier d'en-tête et celui des données brutes. Ce fichier est une compression des données, il permet donc de gagner de l'espace disque. Cependant, il ne permet par de consulter rapidement les caractéristiques de l'image dans un fichier texte indépendant.

2.5 Conclusion

Dans ce chapitre, nous avons introduit les grands principes de l'imagerie médicale puisque nous en parlerons tout au long de ce mémoire. Nous savons maintenant qu'il existe différentes techniques d'acquisition d'images qui donnent diverses informations, ainsi que plusieurs formats d'enregistrement de ces images. Dans le chapitre suivant, nous allons un pas plus loin en présentant la notion de recalage d'images médicales.

Chapitre 3

Le recalage d'images

En traitement d'images, le *recalage* est une technique qui consiste en la *mise en correspondance* d'images, afin de pouvoir comparer ou combiner leurs informations respectives. Cette mise en correspondance se fait par la recherche d'une transformation géométrique permettant de passer d'une image, dite "mouvante", à l'autre, dite "fixe" [53]. Comme nous l'avons expliqué dans le Chapitre 2, ces images peuvent être obtenues grâce à différentes *méthodes d'acquisition*. Elles sont donc de *modalités* différentes, mais elles peuvent également représenter un même patient ou des patients différents.

3.1 Recalage médical

Comme nous l'avons mentionné dans la Section 2.1.1, les différentes techniques d'acquisition d'images ne donnent pas les même informations sur le corps du patient et il est donc intéressant de combiner les informations obtenues par différentes méthodes d'acquisition, afin d'enrichir l'information. Nous utilisons donc le recalage d'images pour mettre en commun les données issues de plusieurs techniques, par exemple pour combiner des données anatomiques avec des données fonctionnelles. C'est le *recalage multimodal*, puisque les images à recaler proviennent de modalités différentes. Cependant, il est parfois nécessaire de recaler des images de la même modalité, nous parlerons alors de *recalage monomodal*.

Nous avons fait la distinction entre deux types de recalage d'images, le recalage monomodal et multimodal. De plus, il est parfois nécessaire de recaler deux images représentant un même patient ou deux images représentant des patients différents. Nous parlons de *recalage intrasujet* lorsque nous voulons recaler deux images d'un même patient et de *recalage intersujets* lorsque nous voulons recaler deux images de patients différents. Finalement, nous distinguons quatre types de recalage : le recalage multimodal/intrasujet, multimodal/intersujets, monomodal/intrasujet et monomodal/intersujets.

3.2 Exemples de recalage

3.2.1 Le recalage multimodal

Commençons par illustrer le recalage multimodal/intrasujet avec la Figure 3.1, où nous avons trois images de modalités différentes et représentant le corps humain d'un même patient.

L'image de gauche, obtenue par un CT-scan, montre l'anatomie du corps humain avec une boule anormale dans un poumon. L'image centrale, obtenue par un PET-scan, indique que la boule est probablement cancéreuse à cause de l'augmentation de l'activité cellulaire. L'image de droite est la combinaison, le recalage, des deux images, qui permet de profiter des caractéristiques anatomiques de l'une (CT-scan) et des caractéristiques fonctionnelles de l'autre (PET-scan).

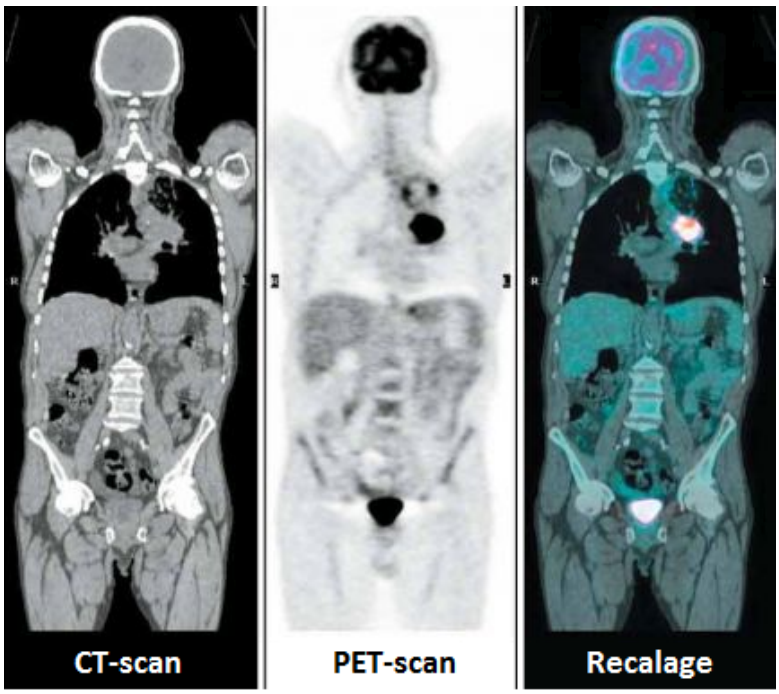


FIGURE 3.1 – Images d'un même patient obtenues par CT-scan et PET-scan [41]

Pour ce premier exemple de recalage multimodal/intrasujet, l'information anatomique est fournie par un CT-scan. Donnons maintenant une illustration du recalage multimodal/intrasujet où l'information anatomique est fournie par une IRM. La Figure 3.2 illustre différentes images du cerveau d'un même patient. L'image de gauche, obtenue par une IRM, donne l'anatomie du cerveau. L'image centrale, obtenue par un PET-scan, indique la présence d'une tumeur cérébrale primitive. L'image de droite est la combinaison, le recalage, des deux images qui permet de connaître la localisation (indiquée par la flèche) de la tumeur en utilisant les caractéristiques anatomiques de l'IRM et les caractéristiques fonctionnelles du PET-scan. Ce type de recalage permet donc d'associer une activité cérébrale à une structure, ainsi que d'évaluer les caractéristiques de la tumeur avant et au cours d'un traitement. Ceci est indispensable pour choisir le traitement le mieux adapté.

Comme le recalage multimodal/intersujets est peu utilisé en médecine nucléaire, nous ne l'illustrons pas dans le cadre de ce mémoire.

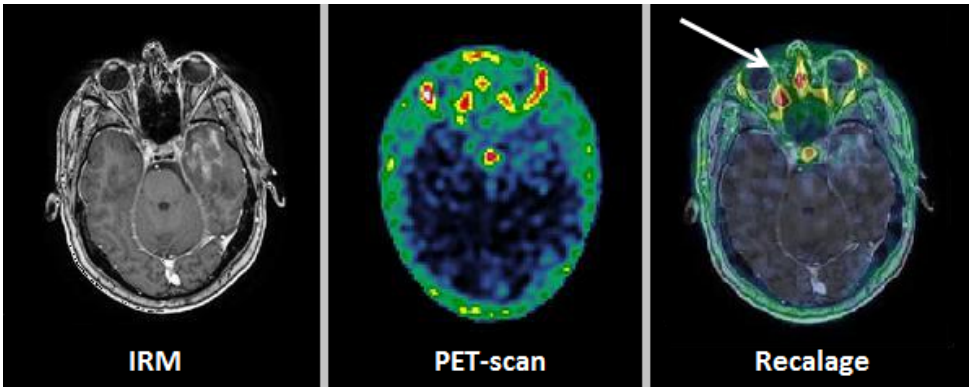


FIGURE 3.2 – Images d'un même patient obtenues par IRM et PET-scan [9]

3.2.2 Le recalage monomodal

Le recalage monomodal/intrasujet est utilisé pour mettre en correspondance des images d'un même patient prises à des moments différents pour suivre l'évolution d'une lésion ou d'une pathologie. Dans ce cas, le recalage est utilisé pour compenser les changements de position du patient d'un examen à l'autre. La Figure 3.3 illustre l'évolution d'une lésion d'un patient atteint de sclérose en plaques. Les images a) et b) représentent des images IRM de ce patient à quelques mois d'intervalle. Afin d'analyser l'évolution de la maladie, il faut faire correspondre l'image source a) avec la cible b). Pour ce faire, nous utilisons un recalage monomodal dont le résultat se trouve sur l'image c) qui est la source recalée. Pour finir, l'image d) est la différence finale entre l'image source recalée et l'image cible et montre clairement que la maladie s'est amplifiée.

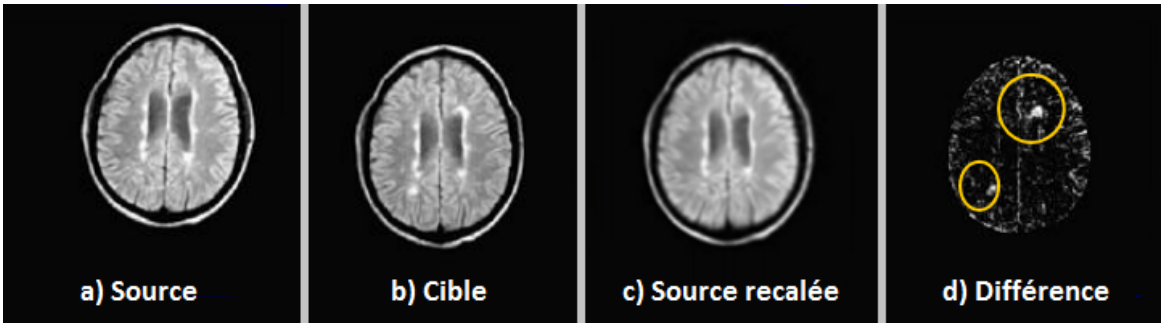


FIGURE 3.3 – Images d'un même patient obtenues par IRM [30]

Le recalage monomodal/intersujets, quant à lui, met en correspondance des informations entre plusieurs patients, ce qui permet de créer des modèles anatomiques ou fonctionnels de référence. En d'autres mots, un tel recalage permet de créer un *atlas*. Il permet également de superposer des données cliniques avec un atlas, ainsi que d'effectuer d'éventuelles études statistiques sur une population. La Figure 3.4 illustre le recalage d'une image IRM d'un individu afin de pouvoir être comparée avec l'IRM de l'atlas. L'image de gauche, obtenue par IRM, représente le cerveau d'un patient. L'image centrale, également obtenue par IRM, est le cerveau "de référence" qui constitue l'atlas. L'image de droite est le recalage du cerveau du patient qui peut alors être comparée à l'atlas.

FIGURE 3.4 – Images de plusieurs patients obtenues par IRM [30]

3.3 Attributs de l'image

Les méthodes de recalage peuvent être classées selon les *attributs de l'image*. Dans ce cas, les méthodes de recalage d'images se basent sur plusieurs types d'informations. Pour estimer la transformation, on peut se baser sur des primitives géométriques ou sur l'intensité des images. Il existe donc deux approches : l'*approche géométrique* et l'*approche iconique*.

3.3.1 Approche géométrique

Les méthodes de recalage basées sur l'approche géométrique furent parmi les premières à être proposées de par leur similitude avec la méthode utilisée par un être humain pour mettre en correspondance deux images [58]. Ces méthodes se basent sur l'*extraction de caractéristiques (ou primitives) géométriques* de chacune des images. Ces caractéristiques géométriques peuvent être des points, des coins, des contours et bien d'autres. Avec ce type d'informations, le recalage consiste à déterminer une transformation qui met en relation la position d'une caractéristique dans une image (*l'image "mouvante"*) avec la position de la même caractéristique dans l'autre image (*l'image "fixe"*). Nous cherchons une transformation de la forme :

$$T : \Omega_M \to \Omega_F : \mathrm{x} \to T(\mathrm{x}),$$

où Ω_M et Ω_F sont, respectivement, les domaines de l'image mouvante et de l'image fixe et x est une position dans le domaine de l'image mouvante. Cette transformation est telle que l'écart entre les deux images soit minimal et donc que la distance entre des caractéristiques homologues soit minimale au sens des moindres carrés. Ce concept de problème aux moindres carrés est expliqué dans la Section 4.4.3 du chapitre suivant.

3.3.2 Approche iconique

L'approche iconique du recalage se fonde sur la *comparaison locale des intensités*. Les primitives utilisées ne sont pas, contrairement aux méthodes de l'approche géométrique, de nature géométrique, mais correspondent à des vecteurs à quatre dimensions contenant la position et l'intensité. Elles ne nécessitent aucune extraction de caractéristiques géométriques et donc aucune compréhension de la structure géométrique de l'image. L'approche iconique est plus adaptée au recalage multimodal des images médicales [1].

L'approche iconique désigne donc le recalage qui se base sur les intensités, le niveau de gris des deux images via une mesure de similarité, une métrique. Il existe différentes mesures de similarité. La métrique la plus utilisée est la somme du carré des différences (SDD). Dans ce cas, nous ajustons une transformation de la forme :

$$T : \Omega_M \to \Omega_F : \mathrm{x} \to T(\mathrm{x}),$$

jusqu'à obtenir des intensités identiques. Comme dans l'approche géométrique, Ω_M et Ω_F sont, respectivement, les domaines de l'image mouvante et de l'image fixe et x est une position dans le domaine de l'image mouvante.

3.3.3 Propriétés des méthodes

Les approches vues ci-dessus ont chacune des avantages et des inconvénients comme le montre la Table 3.1. Par exemple, l'approche iconique nécessite un temps de traitement important alors que l'approche géométrique ne demande qu'un temps de traitement réduit.

Méthode	Géométrique	Iconique
Avantages	- Temps de traitement réduit - Caractéristiques très informatives	- Méthode automatique - Adapté au recalage multimodal - Pas de pré-traitement requis
Inconvénients	- Sélection manuelle ou semi-manuelle des caractéristiques - Généralement réservé au recalage monomodal - Pré-traitement nécessaire (segmentation)	- Temps de traitement important - Caractéristiques peu informatives

TABLE 3.1 – Avantages et inconvénients des méthodes de recalage

Pour contourner les inconvénients de ces deux méthodes, nous pouvons combiner les deux méthodes via une dernière approche, appelée l'approche hybride. Dans ce cas, la méthode se base sur des primitives géométriques des images et considère une mesure de similarité de l'approche iconique.

3.4 Modèles de transformation

Dans la section précédente, nous avons présenté le cadre théorique du recalage des images médicales. Deux approches principales ont été présentées : l'approche géométrique et l'approche iconique. En revanche, un point crucial du recalage des images n'a toujours pas été abordé : le *modèle de déformation*, le type de la transformation T. Dans cette section, nous présentons brièvement les différentes classes de transformations ainsi que leur implication dans le recalage des images médicales [1].

Les transformations peuvent être de type *global* ou *local*. Dans le premier cas, la déformation s'applique de manière identique à toute l'image. Dans le deuxième cas, la déformation peut

varier localement dans l'image et donc des sous-sections de l'image ont leur propre transformation.

3.4.1 Approche linéaire

Les transformations linéaires, qui peuvent être rigides ou affines comme nous l'expliquons dans la suite de cette section, se définissent de la manière suivante :

$$T : \mathrm{x} \mapsto A\mathrm{x} + b,$$

où $\mathrm{x} = (x, y, z) \in \mathbb{R}^3$, A est une matrice de dimension 3×3 $\left(A \in \mathbb{R}^{3\times3}\right)$ et $b \in \mathbb{R}^3$ [1]. Nous pouvons également définir cette transformation avec des coordonnées homogènes par :

$$T : \mathrm{x} \mapsto M\dot{\mathrm{x}},$$

où $\dot{\mathrm{x}} = (x, y, z, 1) \in \mathbb{R}^4$, M est une matrice de dimension 4×4 $\left(M \in \mathbb{R}^{4\times4}\right)$ et $M = [A|b]$ [37].

Lorsque nous appliquons plusieurs transformations successives pour que l'image mouvante corresponde à l'image fixe, la transformation totale T est la composition des différentes transformations. Si nous utilisons deux transformations T_1 et T_2, nous avons donc

$$\begin{aligned} T_2(T_1(\mathrm{x})) &= (T_2 \circ T_1)(\mathrm{x}) \\ &= M_2 M_1 \mathrm{x}, \end{aligned}$$

où M_1 et M_2 sont, respectivement, les matrices de transformations de T_1 et T_2.

La transformation la plus simple à envisager est la transformation rigide. Cette transformation est a priori appropriée au recalage d'images monomodal/intrapatient. Une transformation rigide est une isométrie conservant l'orientation. Une transformation rigide est donc tout simplement la composée d'une *translation* suivant x, y ou z et d'une *rotation* suivant x, y ou z. Cette transformation présente l'avantage de conserver les angles et les formes. Par exemple, l'image d'un tétraèdre est un tétraèdre de même nature. Les matrices de transformations correspondant à la translation et à la rotation sont :

$$M_{\text{translation}} = \begin{pmatrix} 1 & 0 & 0 & t_x \\ 0 & 1 & 0 & t_y \\ 0 & 0 & 1 & t_z \\ 0 & 0 & 0 & 1 \end{pmatrix},$$

et

$$\begin{aligned} M_{\text{rotation}} &= R_X R_Y R_Z \\ &= \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\theta & -\sin\theta & 0 \\ 0 & \sin\theta & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} \cos\gamma & 0 & \sin\gamma & 0 \\ 0 & 1 & 0 & 0 \\ -\sin\gamma & 0 & \cos\gamma & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} \cos\delta & -\sin\delta & 0 & 0 \\ \sin\delta & \cos\delta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}, \end{aligned}$$

où t_x, t_y et t_z sont les paramètres de la translation et θ, γ et δ sont les paramètres de la rotation.

Cette transformation rigide peut donc être utilisée uniquement pour recaler deux images ne présentant pas d'évolution de structures, c'est-à-dire pas de croissance, ni de modification du volume d'une sous-structure. La Figure 3.5 illustre un exemple de transformation rigide globale et un exemple de transformation rigide locale.

FIGURE 3.5 – Exemple de transformation rigide [30]

La deuxième transformation linéaire est la transformation affine. Cette transformation est employée à la fois pour des recalages d'images multimodales et pour des recalages intersujets. Une transformation affine est une isométrie qui transforme des lignes parallèles en lignes parallèles. Dans les transformations affines, nous retrouvons donc les transformations rigides (translation et rotation) mais également le *zoom* (le grossissement ou l'homothétie) et le *cisaillement*. Si l'on considère un parallélépipède rectangle, le cisaillement est une variation de l'angle, qui n'est plus droit. Cela correspond à des forces s'exerçant parallèlement à la face [51]. Afin d'appliquer un zoom ou un cisaillement à l'image mouvante, il faut appliquer les matrices suivantes :

$$M_{\text{zoom}} = \begin{pmatrix} s_x & 0 & 0 & 0 \\ 0 & s_y & 0 & 0 \\ 0 & 0 & s_z & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix},$$

et

$$M_{\text{cisaillement}} = \begin{pmatrix} 1 & s_{yx} & s_{zx} & 0 \\ -s_{yx} & 1 & s_{zy} & 0 \\ -s_{zx} & -s_{zy} & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix},$$

où s_x, s_y et s_z sont les paramètres du zoom et s_{yx}, s_{zx} et s_{zy} sont les paramètres du cisaillement. La Figure 3.6 illustre un exemple de transformation affine globale et un exemple de transformation affine locale.

3.4.2 Approche non linéaire

Les transformations linéaires ne sont pas suffisantes pour recaler toutes les images. Il est nécessaire d'utiliser des transformations non rigides comme pour la mise en correspondance entre un atlas du cerveau et les données CT-scan d'un sujet humain. Dans le cas de transformations non rigides, toutes les déformations sont possibles. Nous appliquons dès lors l'ensemble des transformations vues ci-dessus mais *localement*, par petites régions. La Figure 3.7 illustre

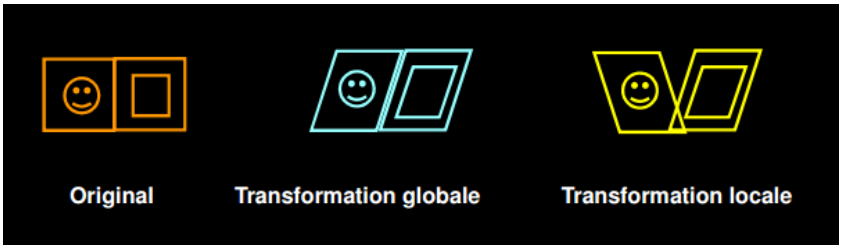


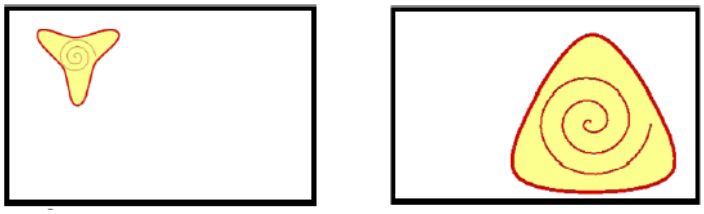
FIGURE 3.6 – Exemple de transformation affine [30]

un exemple de transformation non rigide globale et un exemple de transformation non rigide locale.

FIGURE 3.7 – Exemple de transformation non-rigide [30]

3.4.3 Exemple de transformation

Pour illustrer les différentes transformations possibles, reprenons l'exemple de S. Calande [7] où nous voulons recaler l'image mouvante (Figure 3.8 (a)) sur l'image fixe (Figure 3.8 (b)).

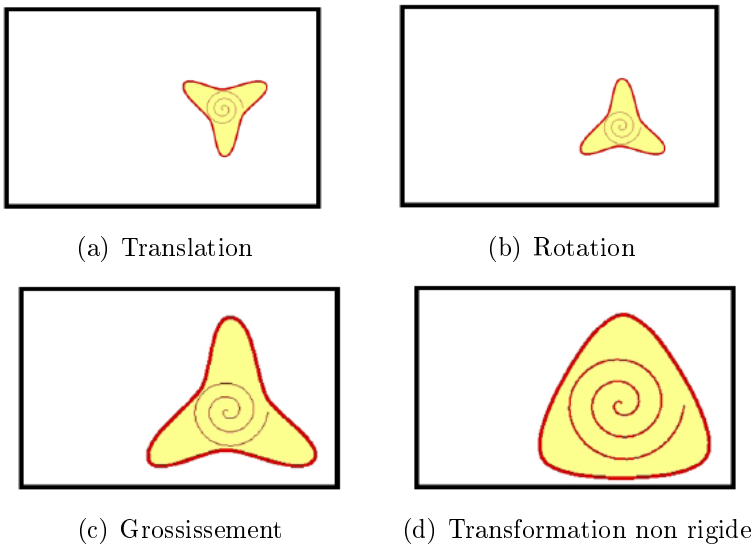


(a) Image mouvante (b) Image fixe

FIGURE 3.8 – Images de départ de l'exemple de recalage [7]

Pour que l'image mouvante soit recalée sur l'image fixe, nous devons appliquer les trois types de méthodes vues ci-dessus. Premièrement, nous appliquons un recalage rigide en faisant une translation de l'image mouvante selon l'axe y (translation horizontale) et selon l'axe z (translation verticale), comme illustré sur la Figure 3.9 (a). De cette manière, l'image mouvante se trouve à l'emplacement de l'image fixe. Deuxièmement, nous appliquons encore un recalage rigide avec une rotation par rapport à l'axe y (Figure 3.9 (b)). Troisièmement, nous faisons une transformation affine avec un grossissement (Figure 3.9 (c)). Et quatrièmement, nous utilisons

le recalage non linéaire, et plus particulièrement non rigide, avec des translations locales (voir Figure 3.9 (d)).



(a) Translation (b) Rotation

(c) Grossissement (d) Transformation non rigide

FIGURE 3.9 – Les différents recalages utilisés [7]

3.5 Exemple récapitulatif

Utilisons l'exemple intuitif présenté par G. Malandain [24], pour se familiariser avec l'ensemble des concepts cités ci-dessus. Soient X et Y, deux images d'une même scène, acquises par un capteur (Figure 3.10). Supposons que les deux acquisitions aient été effectuées à des instants différents, et qu'entre-temps la position du capteur ait changé. L'opération du recalage consiste à trouver la transformation géométrique $T(.)$ qui relie les coordonnées x, évoluant dans l'espace de l'image mouvante X, aux coordonnées y, appartenant à l'image fixe Y, telle que :

$$\mathrm{y} \quad = \quad T(\mathrm{x}).$$

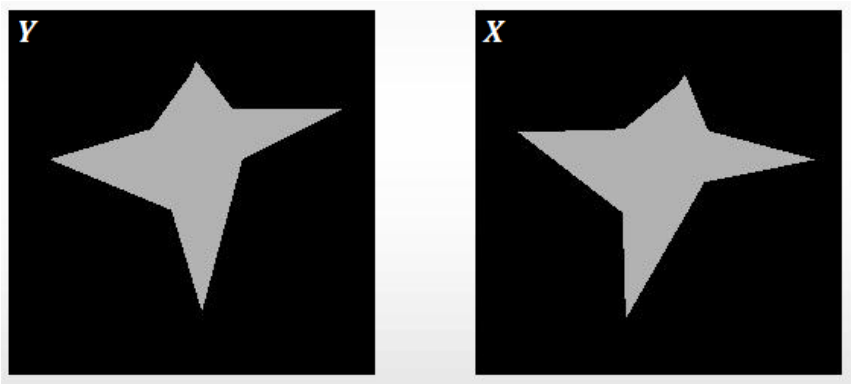


FIGURE 3.10 – Images de départ de l'exemple récapitulatif de recalage [24]

3.5.1 Approche géométrique

Pour estimer la transformation, basons-nous, dans un premier temps, sur des primitives géométriques des deux images. Nous prenons comme caractéristique géométrique les points de forte courbure. Avant d'aller plus loin, expliquons la notion de forte et faible courbure. Pour ce faire, considérons la fonction de la Figure 3.11 comme le tracé d'une route [18]. L'automobiliste roulant sur cette route doit fortement braquer aux points A, D, E et G, alors qu'il n'aura presque pas besoin de braquer aux points B, C, F et H. On dit que la courbe possède une forte courbure en A, D, E et G et une faible courbure en B, C, F et H. De la même manière, les points A, D, E et G sont des points de forte courbure et les points B, C, F et H sont des points de faible courbure.

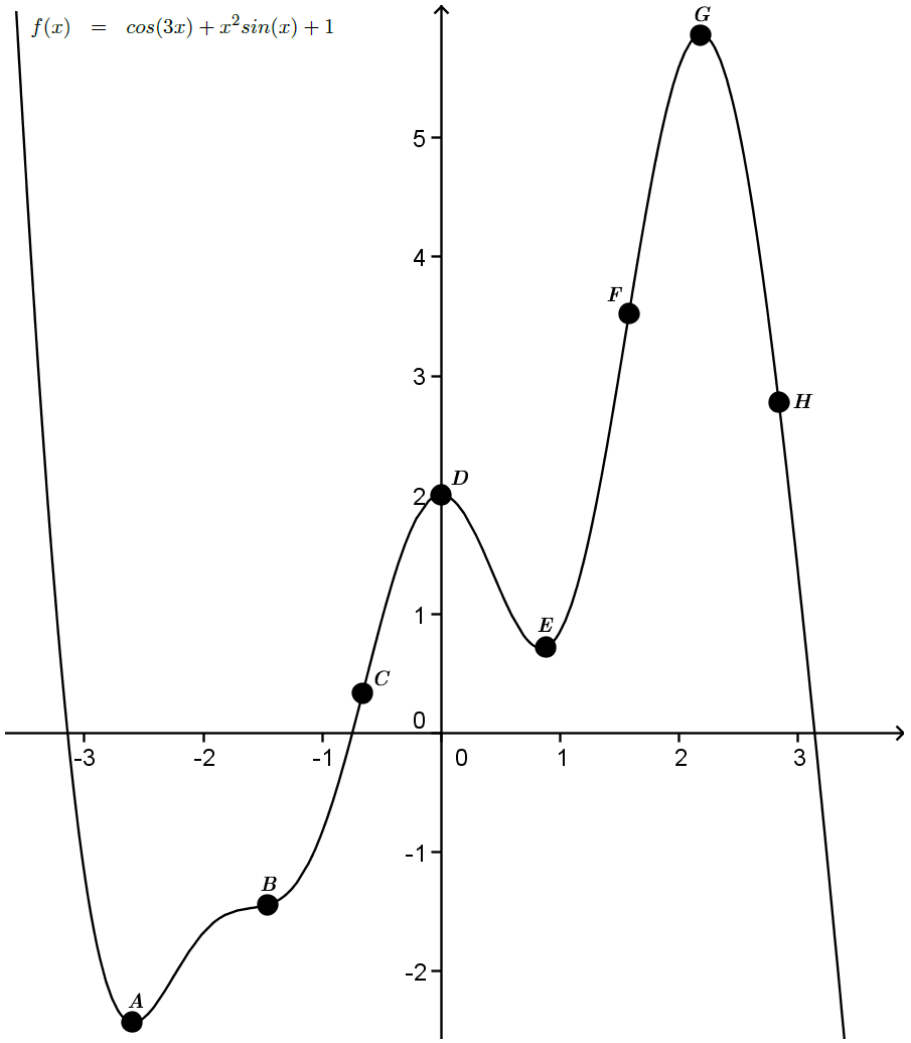


FIGURE 3.11 – Notion de courbure [24]

Grâce à cette notion de courbure, nous pouvons revenir à notre exemple et détecter l'ensemble des points de forte courbure de l'image mouvante et de l'image fixe. La Figure 3.12 indique les 8 paires de points de forte courbure de correspondance extraites des deux images :

$$X = \{x_i\},$$

$$Y = \{y_i\},$$

avec $i = 1, ..., 8$. Les paramètres de la transformation géométrique sont trouvés en minimisant une distance quadratique, notée S, entre chaque couple de points de forte courbure dans le sens des moindres carrés (voir Section 4.4.3) :

$$S(T) = \sum_{i=1}^{8} ||T(x_i) - y_i||^2,$$

où, rappelons-le, $\{x_1, ..., x_8\}$ est l'ensemble des points de forte courbure de l'image mouvante X et $\{y_1, ..., y_8\}$ est l'ensemble des points de forte courbure de l'image fixe Y. Nous voulons donc minimiser la distance entre les points $T(x_i)$ et y_i, avec $i = 1, ..., 8$, comme illustré sur la Figure 3.13. Afin de minimiser la mesure de similarité S, il suffit d'appliquer une transformation rigide, plus particulièrement une rotation dans le sens inverse des aiguilles d'une montre à l'image mouvante comme nous l'avons appliqué sur la Figure 3.14.

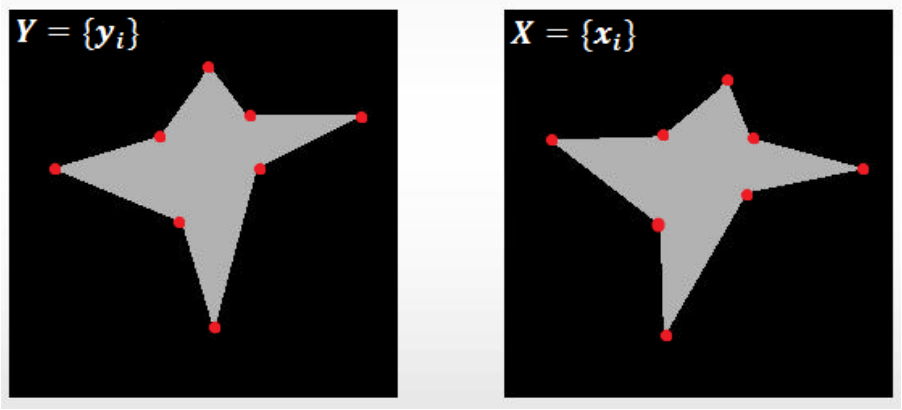


FIGURE 3.12 – Points de fortes courbures à faire correspondre [24]

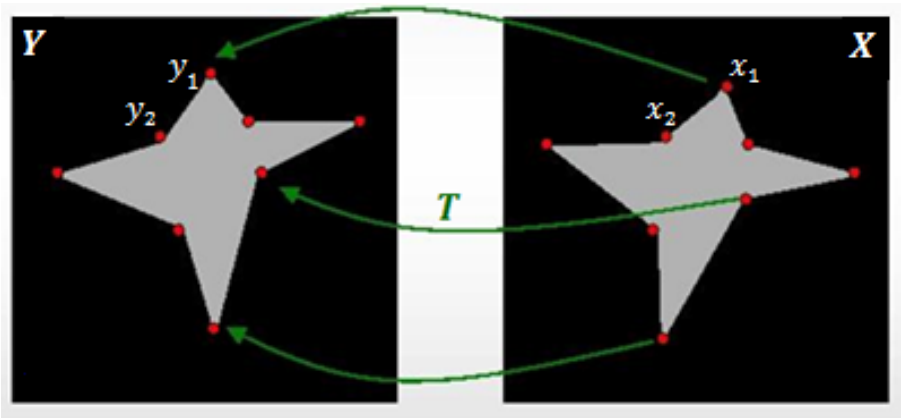


FIGURE 3.13 – Minimisation de la mesure entre les points [24]

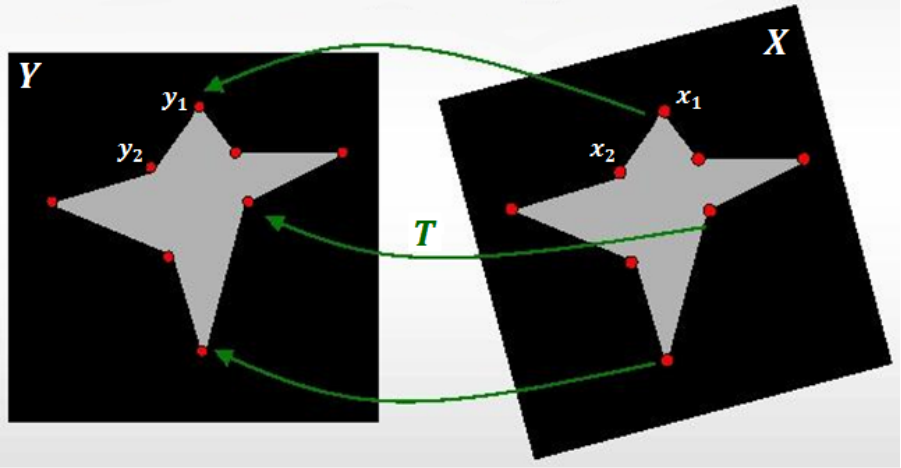


FIGURE 3.14 – Rotation de l'image mouvante [24]

3.5.2 Approche iconique

Nous utilisons maintenant une approche iconique pour trouver la transformation nécessaire afin de recaler ces deux images. Notons $I(x)$ l'intensité au point x où nous faisons l'hypothèse que les deux images sont exprimées dans la même échelle d'intensité. Nous prenons 6 points d'intensités différentes sur l'image mouvante et nous devons minimiser la métrique qui est donnée par la somme du carré de la différence des intensités des images :

$$S(T) \quad = \quad \sum_{k=1}^{6} \left(I(x_k) - I(T(x_k))\right)^2 .$$

Pour commencer, nous devons minimiser la différence d'intensité entre les points homologues x_1 et $T(x_1)$ comme illustré sur la Figure 3.15.

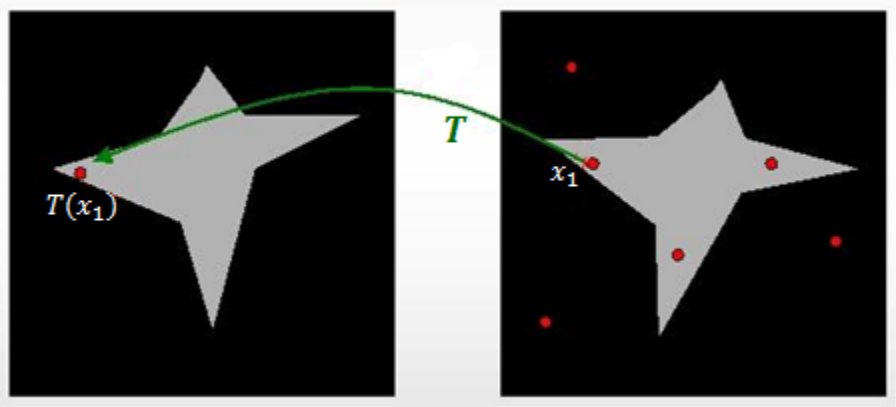


FIGURE 3.15 – Points de même intensité à recaler [24]

Pour ce faire, nous appliquons une première transformation T_1 (Figure 3.16) qui est une simple translation, ou avec d'autres mots, la transformation identité. Sur cette figure, nous

distinguons les zones identiques aux deux images et les zones où il y a une différence d'intensité entre l'image mouvante et l'image fixe. Ensuite, nous appliquons une deuxième transformation T_2 (Figure 3.17) qui est, dans ce cas, une rotation dans le sens inverse des aiguilles d'une montre. De cette manière, les points de même intensité x_1 et $T(x_1)$ sont superposés.

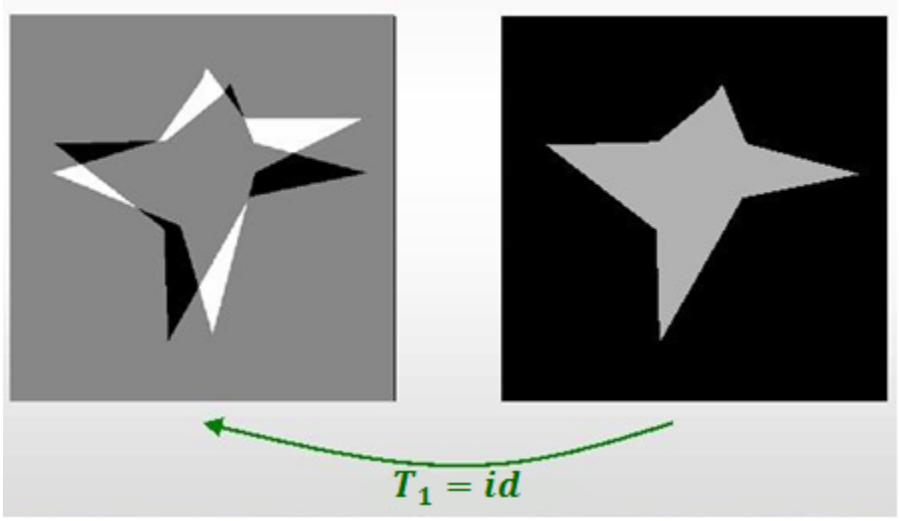


FIGURE 3.16 – Première transformation : identité [24]

La deuxième transformation de la Figure 3.17 donne lieu à un recouvrement partiel des images. En effet, nous minimisons l'intensité pour un seul point de l'image mouvante. Cependant, pour assurer un bon résultat, nous devons faire la même démarche pour plusieurs points d'intensités différentes.

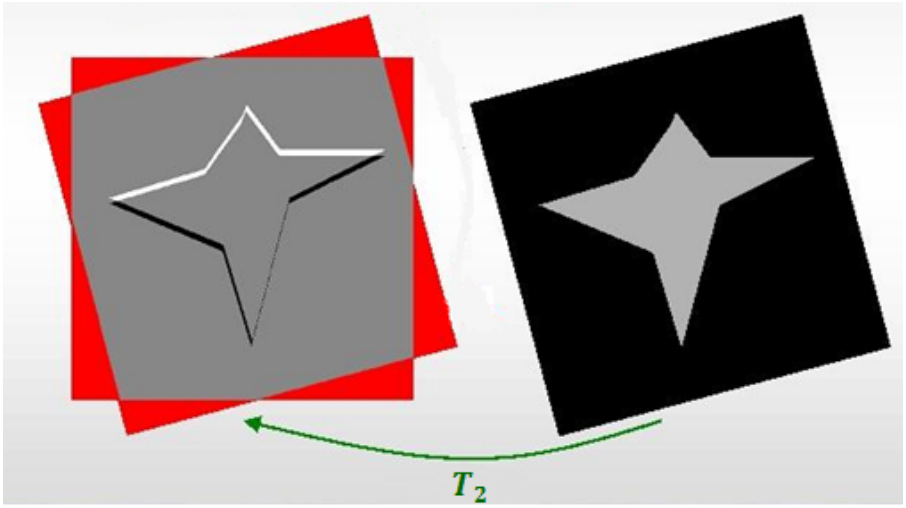


FIGURE 3.17 – Deuxième transformation : rotation [24]

3.6 Interpolation

Dans le cadre du recalage d'images par approche iconique, il est nécessaire de calculer les valeurs d'intensité du point obtenu après avoir appliqué la transformation. Lorsque l'image mouvante est transformée une première fois, il faut définir un grillage similaire à celui de l'image fixe avant d'appliquer la deuxième transformation. Ce grillage permet de calculer les intensités de chaque coordonnée. Comme les déplacements des pixels ne sont pas toujours proportionnels au grillage, l'*interpolation* ou le *rééchantillonnage* est primordiale.

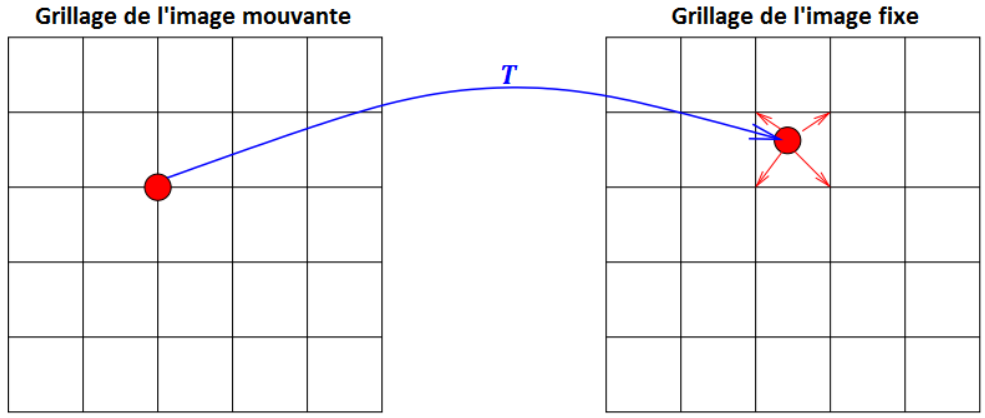


FIGURE 3.18 – Transformation directe [5]

La technique la plus utilisé dans la littérature est la méthode d'interpolation linéaire basée sur les plus proches voisins. Dans ce cas, la valeur du pixel le plus proche est affecté au point considéré. Selon L. Risser [34], d'autres méthodes existent comme l'interpolation bilinéaire ou trilinéaire, l'interpolation suivant une gaussienne ou suivant le sinus cardinal. Pour faciliter les opérations de calcul, nous ne déplaçons pas l'image mouvante selon des transformations locales ou globales pour ensuite interpoler toutes les intensités sur le grillage de l'image fixe comme illustré sur la Figure 3.18. Nous travaillons par transformation inverse puis interpolation sur un grillage bougeant sur l'image mouvante comme indiqué par la Figure 3.19.

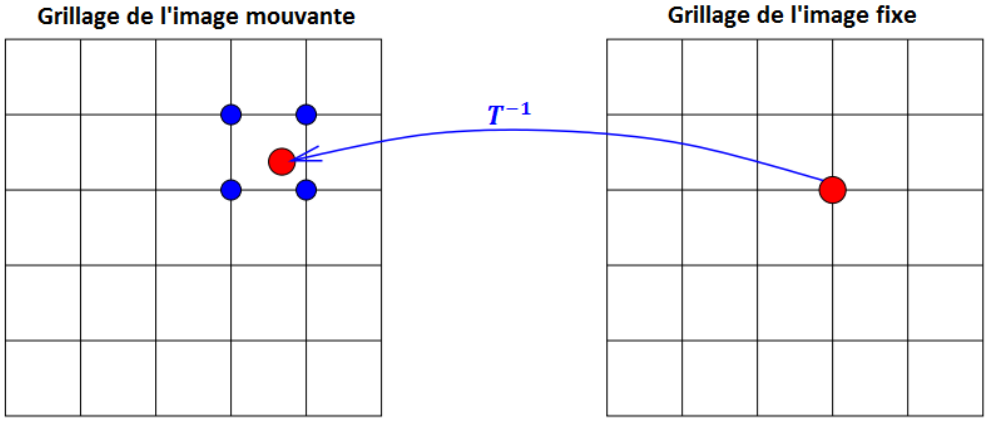


FIGURE 3.19 – Transformation indirecte [5]

Le problème est de partir d'une position de pixel entière dans l'image mouvante et de tomber sur une position de pixel fractionnaire dans l'image mouvante, il faut alors interpoler les pixels. Pour chaque position de l'image fixe, on calcule grâce à la transformation inverse T^{-1} la position dans l'image mouvante. On interpole alors pour trouver la valeur de cette position à partir des pixels voisins. Nous attribuons donc à cette position, la valeur du pixel voisin le plus proche.

3.7 Optimisation

Comme mentionné par L. Risser [34], le problème du recalage d'images est très complexe. La fonction à maximiser ou à minimiser est une fonction de ressemblance entre deux images. Dans le cadre de transformations rigides, cette fonction ne dépend pas d'un, mais de six paramètres (3 pour la translation et 3 pour la rotation). Lors de transformations affines, cette fonction peut dépendre de douze paramètres (3 pour la translation, 3 pour la rotation, 3 pour le zoom et 3 pour le cisaillement).

Les *méthodes d'optimisation* constituent donc le lien entre le critère de ressemblance et la transformation optimisée, utilisant les informations issues du ou des critères choisis afin de déduire la transformation entre les images. Les algorithmes de recalage dépendent de la transformation optimisée et du critère. Les méthodes géométriques produisent souvent un ensemble d'appariements entre divers points des images. Dans ce cas, et si la transformation recherchée le permet, un algorithme des moindres carrés (voir Section 4.4.3) peut être utilisé afin d'obtenir la transformation optimale. Les critères iconiques sont quant à eux souvent optimisés via des méthodes de descente de gradient (voir Section 4.4.1) en fonction des paramètres de la transformation ou, lorsque le gradient ne peut être calculé, par la méthode de Powell [28] afin d'optimiser la transformation.

Dans le chapitre suivant, nous introduisons l'optimisation de manière générale, ainsi que les méthodes d'optimisation qui seront utilisées dans la suite de ce mémoire.

Chapitre 4

Optimisation

Dans ce chapitre, nous introduisons d'abord l'*optimisation avec et sans contraintes*. Ensuite, nous présentons les *méthodes d'optimisation sans contraintes* qui sont utilisés dans la suite de ce mémoire. Pour ce faire, nous nous basons sur les ouvrages de M. Bierlaire [3] et de J. Nocedal et S. J. Wright [28].

4.1 Introduction

De manière générale, l'optimisation est présente dans de nombreux domaines d'application. Les investisseurs cherchent à atteindre le taux de rendement le plus élevé possible. Les fabricants visent une efficacité maximale dans la conception et le fonctionnement de leurs processus de production. Les molécules d'un système chimique réagissent les unes avec les autres jusqu'à ce que l'énergie potentielle totale des électrons soit réduite au minimum. Les rayons de lumière suivent des chemins qui minimisent leur temps de parcours.

L'optimisation est un outil important en science décisionnelle et en analyse des systèmes physiques. Dans le but de l'utiliser, il faut d'abord identifier une *fonction objective*, une mesure quantitative de la performance du système à étudier. Cette fonction peut être le profit, le temps, l'énergie potentielle, ou n'importe quelle quantité qui dépend de certaines caractéristiques du système, appelées variables ou inconnues. Notre objectif est de trouver les valeurs des variables qui optimisent la fonction objective. Souvent, les variables sont contraintes. Par exemple, le taux d'intérêt sur un prêt ne peut pas être négatif.

Le processus d'identification de la fonction objective, des variables et des contraintes pour un problème donné est la *modélisation*. La construction d'un modèle approprié est la première étape du processus d'optimisation. Une fois que le modèle est formulé, un *algorithme d'optimisation* peut être utilisé pour trouver la solution. Il existe de nombreux algorithmes, chacun est adapté à un type particulier de problème d'optimisation. Le choix de l'algorithme est important car il détermine si la solution est trouvée lentement ou rapidement. En effet, il existe des algorithmes à convergence très efficace et robuste.

4.2 Formulation mathématique

Mathématiquement, l'*optimisation* consiste à trouver le minimum ou le maximum d'une fonction soumise à des contraintes sur ses variables. Nous utilisons les notations suivantes :

- x est le vecteur des variables, également appelées inconnues ou paramètres ;
- f est la fonction objective, une fonction de x que l'on veut maximiser ou minimiser ;
- c est le vecteur de contraintes auxquelles doivent satisfaire les inconnues. Ce vecteur contient des fonctions de la variable x.

Par exemple, le minimum de $f(x) = (x-1)^2$ pour $x \in \mathbb{R}$ est, de manière triviale, égal à 1. Cependant, en général, les fonctions à étudier sont bien plus compliquées.

Parmi les problèmes d'optimisation, il y a les problèmes d'optimisation sans contraintes (4.1) et les problèmes d'optimisation avec des contraintes d'égalité et/ou d'inégalité (4.2) :

$$\min_{x \in \mathbb{R}^n} \quad f(x), \tag{4.1}$$

et

$$\begin{cases} \min_{x \in \mathbb{R}^n} & f(x) \\ s.c. & c_i(x) = 0, \quad i \in \Omega, \\ & c_i(x) \geqslant 0, \quad i \in \Theta, \end{cases} \tag{4.2}$$

où les fonctions $f : \mathbb{R}^n \to \mathbb{R}$, $c_i : \mathbb{R}^n \to \mathbb{R}^m$ pour $i \in \Omega$ et $c_i : \mathbb{R}^n \to \mathbb{R}^p$ pour $i \in \Theta$ sont supposées non linéaires et deux fois continûment différentiables. Ω et Θ sont des ensembles d'indices qui définissent le nombre de restrictions que nous plaçons sur les variables. De plus, n spécifie la taille du problème et est souvent grand.

Définition 4.2.1 *Une contrainte d'inégalité $c_i(x) \geqslant 0$, $i \in \Theta$, est active en $x^* \in \mathbb{R}^n$ lorsque $c_i(x^*) = 0$.*

Prenons comme exemple illustratif une fonction à deux variables à minimiser avec deux contraintes d'inégalité. Ce problème est illustré sur la Figure 4.1, où la fonction à minimiser est représentée par ses courbes de niveau, et est décrit en (4.3) :

$$\begin{cases} \min_{x \in \mathbb{R}^2} & (x_1-6)^2 + (x_2-3)^2 \\ s.c. & x_1^2 - 4x_2 \geqslant 0, \\ & x_1 + x_2 \leqslant 6. \end{cases} \tag{4.3}$$

Nous pouvons écrire ce problème sous la forme (4.2) en définissant :

$$f(x) = (x_1-6)^2 + (x_2-3)^2, \quad x = \begin{pmatrix} x_1 \\ x_2 \end{pmatrix},$$

$$c(x) = \begin{pmatrix} c_1(x) \\ c_2(x) \end{pmatrix} = \begin{pmatrix} -x_1^2 + 4x_2 \\ -x_1 - x_2 + 6 \end{pmatrix}, \quad \Theta = \{1, 2\}, \quad \Omega = \emptyset.$$

L'ensemble des points où les deux contraintes sont vérifiées est appelé l'ensemble admissible (en gris sur la Figure 4.1). Le minimum admissible de ce problème est le point x^* où les deux contraintes sont actives.

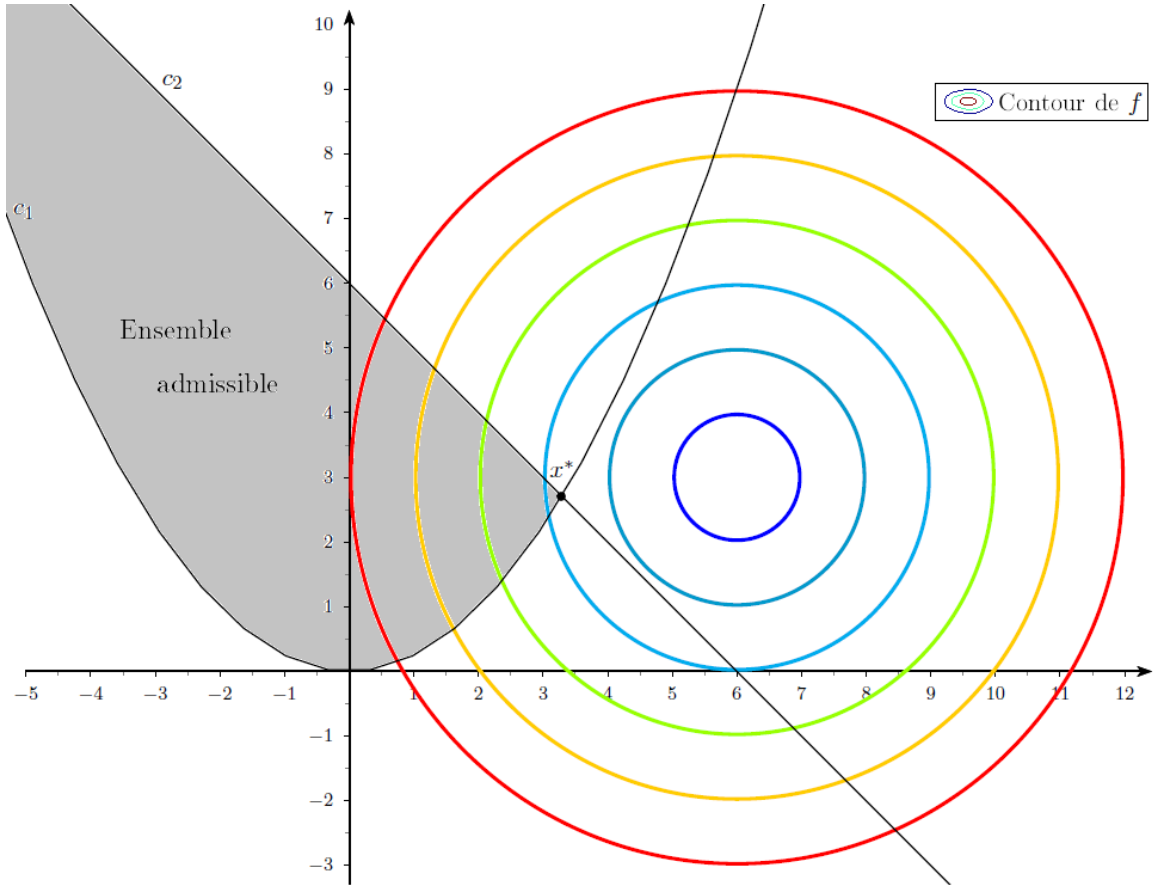


FIGURE 4.1 – Exemple d'optimisation à 2 dimensions - Construit grâce à Matlab [25]

Dans le cas d'un problème sans contraintes, le minimum de cette fonction objective se trouve au centre des courbes de niveau. Dans le cadre de ce mémoire, nous considérons exclusivement des problèmes d'optimisation sans contraintes.

4.3 Conditions d'optimalité sans contrainte

Considérons le problème d'optimisation sans contraintes sous sa forme générale :

$$(P)\begin{cases} \min & f(x), \\ x \in \mathbb{R}^n \end{cases}$$

dans lequel on minimise une fonction $f : \mathbb{R}^n \to \mathbb{R}$ définie sur $\mathbb{R}^n$ et non linéaire (voir [13] par exemple). Rappelons quelques notions relatives au problème (P) qui seront utilisées dans la suite de ce chapitre. L'ensemble $\mathbb{R}^n$ sur lequel on minimise f est l'*ensemble admissible* du problème. Un *minimum global* de ce problème est un point $x^* \in \mathbb{R}^n$ tel que :

$$\forall x \in \mathbb{R}^n \quad , \quad f(x^*) \leq f(x). \tag{4.4}$$

On dit que $x^* \in \mathbb{R}^n$ est un *minimum local* de (P) s'il existe un voisinage $\mathcal{N}$ de x^* tel que :

$$\forall x \in \mathbb{R}^n \cap \mathcal{N} \quad , \quad f(x^*) \leq f(x). \tag{4.5}$$

Les *conditions d'optimalité* sont des équations, des inéquations ou des propriétés que vérifient les solutions de (P) (conditions nécessaires, CN) ou qui assurent à un point d'être solution de (P) (conditions suffisantes, CS). On parlera de conditions du premier ordre lorsque celles-ci ne font intervenir que les dérivées premières de f. Quant aux conditions du second ordre, elles font intervenir les dérivées premières et secondes de cette fonction.

Théorème 4.3.1 ***(Conditions nécessaires d'optimalité)***
Soit x^* un minimum local d'une fonction $f : \mathbb{R}^n \to \mathbb{R}$. Si f est continûment différentiable dans un voisinage ouvert $\mathcal{N}$ de x^*, alors

$$\nabla f(x^*) = 0. \tag{4.6}$$

Si, de plus, f est deux fois continûment différentiable sur $\mathcal{N}$, alors

$$\nabla^2 f(x^*) \text{ est semi-définie positive.} \tag{4.7}$$

La condition (4.6) est appelée condition nécessaire du premier ordre et la condition (4.7) est appelée condition nécessaire du second ordre.

Théorème 4.3.2 ***(Conditions suffisantes d'optimalité)***
Soit une fonction $f : \mathbb{R}^n \to \mathbb{R}$ deux fois continûment différentiable dans un voisinage ouvert $\mathcal{N}$ de $\mathbb{R}^n$ et soit $x^* \in \mathcal{N}$ qui vérifie les conditions suivantes :

$$\nabla f(x^*) = 0, \tag{4.8}$$

et

$$\nabla^2 f(x^*) \text{ est définie positive.} \tag{4.9}$$

Alors x^* est un minimum local de f.

La condition (4.8) est appelée condition suffisante du premier ordre et la condition (4.9) est appelée condition suffisante du second ordre.

Avant de présenter les méthodes d'optimisation, nous introduisons le Théorème de Taylor car il sera souvent utilisé tout au long de ce mémoire.

Théorème 4.3.3 ***(Théorème de Taylor [28])***
Soit $f : \mathbb{R}^n \to \mathbb{R}$ une fonction continûment différentiable et soit $u \in \mathbb{R}^n$. Nous avons alors que :

$$f(x+u) \;=\; f(x) + \nabla f(x+tu)^T u, \tag{4.10}$$

pour un $t \in (0,1)$. De plus, si f est deux fois continûment différentiable, nous avons que :

$$f(x+u) \;=\; f(x) + \nabla f(x)^T u + \frac{1}{2} u^T \nabla^2 f(x+tu) u, \tag{4.11}$$

pour un $t \in (0,1)$.

4.4 Les méthodes d'optimisation

A l'heure actuelle, il existe de nombreux algorithmes permettant de chercher le minimum d'une fonction dans le cadre de l'optimisation sans contraintes. Dans tous ces algorithmes, nous devons fournir un point de départ, désigné par x_0. En partant de ce point de départ, les algorithmes d'optimisation génèrent une suite d'itérés $(x_k)_k$ tels que :

$$f(x_{k+1}) \leq f(x_k), \qquad k = 1, 2, \ldots .$$

Comme nous l'avons écrit dans l'introduction, il n'existe pas d'algorithme universel qui est efficace pour minimiser toutes les fonctions. Chaque type de fonction à minimiser, a certaines méthodes d'optimisation plus adaptées que d'autres. Dans le cadre de ce mémoire, nous nous intéressons exclusivement aux méthodes de *recherche linéaire*.

La stratégie de recherche linéaire consiste à choisir une direction d_k et à rechercher dans cette direction un nouvel itéré x_{k+1} à partir de x_k tel que la valeur de la fonction en x_{k+1} soit inférieure à celle en x_k. Autrement dit, chaque itération calcule une direction de recherche d_k et décide dans quelle mesure il faut se déplacer le long de cette direction :

$$x_{k+1} = x_k + \alpha_k d_k,$$

où α_k est un scalaire positif appelé la longueur du pas. L'idée principale est de suivre une direction telle que $\nabla f(x_k)^T d_k < 0$. Cette expression définit une *direction de descente* pour la fonction f. Pour le vérifier, réécrivons cette expression en utilisant la définition de la dérivée :

$$\begin{aligned} \nabla f(x)^T d &= f'(x, d) \\ &= \lim_{\alpha \xrightarrow[>]{} 0} \frac{f(x + \alpha d) - f(x)}{\alpha}. \end{aligned}$$

Si $\nabla f(x_k)^T d_k < 0$, nous obtenons bien que $f(x + \alpha d) < f(x)$ pour $\alpha > 0$ suffisamment petit. Une telle direction permet donc de "descendre" dans la fonction considérée. De plus, la direction de recherche prend généralement la forme :

$$d_k = -B_k^{-1} \nabla f(x_k), \tag{4.12}$$

où B_k est une matrice symétrique et inversible. Dans le cas de la *méthode de plus forte pente*, B_k est la matrice identité I. Pour la *méthode de Newton*, B_k est la matrice Hessienne exacte, $\nabla^2 f(x_k)$, et dans le cas de la *méthode quasi-Newton*, B_k est une approximation de la matrice Hessienne. La direction d_k définie par (4.12) est une direction de descente lorsque B_k est définie positive puisque

$$\nabla f(x_k)^T d_k = -\nabla f(x_k)^T B_k^{-1} \nabla f(x_k) \tag{4.13}$$

est négatif dans ce cas. Dans les sections qui suivent, nous présentons plus en détails ces trois méthodes puisque nous les utilisons dans les prochains chapitres.

4.4.1 Méthodes de plus forte pente

Comme nous l'avons énoncé auparavant, la méthode de plus forte pente est une méthode de recherche linéaire qui se déplace le long de $d_k = -\nabla f(x_k)$ à chaque étape. Cette direction opposée au gradient est celle dans laquelle la fonction a la plus forte descente puisque cette direction est orthogonale aux contours de la fonction et pointe vers des valeurs décroissantes de la fonction. Une itération de cette méthode consiste en

$$x_{k+1} = x_k - \alpha_k \nabla f(x_k). \tag{4.14}$$

Pour le calcul de la longueur du pas α_k, nous devons faire un compromis entre une réduction suffisante de f et le temps de calcul. Le choix idéal est celui qui diminue le plus la fonction dans la direction d_k :

$$\alpha_k = \underset{\alpha>0}{\operatorname{argmin}} f(x_k + \alpha d_k). \tag{4.15}$$

Cette méthode permettant de calculer la longueur du pas est une recherche linéaire exacte. Illustrons, grâce à l'exemple suivant, cette méthode de plus forte pente.

Exemple 4.4.1 ***(Méthode de la plus forte pente)*** *[3]*
Nous allons minimiser la fonction $f : \mathbb{R}^2 \to \mathbb{R}$ définie par

$$f(x) = \frac{1}{2}x_1^2 + \frac{9}{2}x_2^2,$$

en utilisant la méthode de la plus forte pente. Notons x_k l'itéré courant. La direction de la plus forte pente est donnée par :

$$\begin{aligned} d_k &= -\nabla f(x_k) \\ &= \begin{pmatrix} -(x_k)_1 \\ -9(x_k)_2 \end{pmatrix}. \end{aligned} \tag{4.16}$$

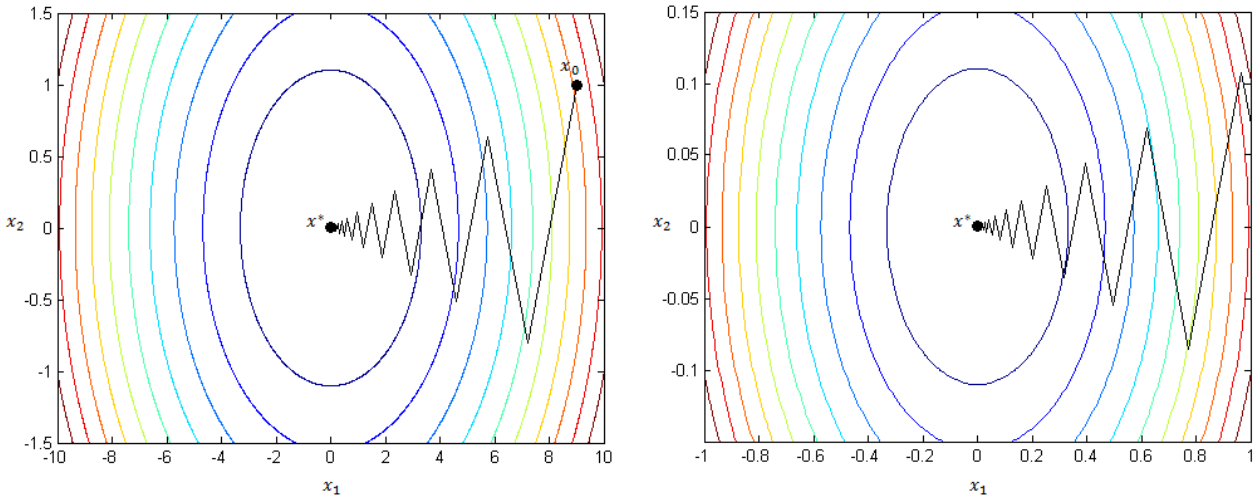
Pour calculer la longueur du pas α_k, nous résolvons le problème à une dimension suivant :

$$\min_{\alpha>0} f(x_k - \alpha \nabla f(x_k)) = \min_{\alpha>0} \frac{1}{2}\left((x_k)_1 - \alpha(x_k)_1\right)^2 + \frac{9}{2}\left((x_k)_2 - 9\alpha(x_k)_2\right)^2.$$

Afin de résoudre ce problème, nous posons $\varphi(\alpha) = f(x_k - \alpha\nabla f(x_k))$ et nous calculons α tel que $\varphi'(\alpha) = 0$. De cette manière, nous trouvons l'expression de la longueur du pas :

$$\alpha_k = \frac{(x_k)_1^2 + 81(x_k)_2^2}{(x_k)_1^2 + 729(x_k)_2^2}. \tag{4.17}$$

La méthode de la plus forte pente génère donc, à chaque itération, le point $x_{k+1} = x_k + \alpha_k d_k$ avec la direction d_k donnée par (4.16) et la longueur du pas α_k donnée par (4.17). En appliquant cet algorithme à partir du point $x_0 = \begin{pmatrix} 9 & 1 \end{pmatrix}^T$, nous obtenons les itérations illustrées sur la Figure 4.2.



(a) Itérations (b) Zoom

FIGURE 4.2 – Exemple de la méthode de plus forte pente - Construit grâce à Matlab [25]

Nous pouvons remarquer, dans cet exemple, que la méthode de la plus forte pente converge extrêmement lentement. Le comportement en zigzag illustré sur la Figure 4.2 est caractéristique de cette méthode. De plus, la résolution du problème (4.15) peut s'avérer coûteuse lorsque la fonction f n'est pas linéaire. Même si ce problème d'optimisation n'a qu'une seule variable, α, il n'est pas trivial à résoudre en général.

Pour contourner ce problème, la technique la plus utilisée est d'effectuer une *recherche linéaire inexacte* afin d'identifier une longueur de pas qui réalise une réduction suffisante de f à un coût minime. Généralement, les algorithmes de recherche linéaire inexacte testent une série de candidats pour α et s'arrêtent lorsque certaines conditions sont satisfaites. Dans notre cas, la longueur du pas doit vérifier les conditions de Wolfe (1969) présentées ci-après.

Définition 4.4.1 ***(Diminution suffisante : première condition de Wolfe [3])*** *Soient $f : \mathbb{R}^n \to \mathbb{R}$ une fonction différentiable, un point $x_k \in \mathbb{R}^n$, un direction de descente $d_k \in \mathbb{R}^n$ telle que $\nabla f(x_k)^T d_k < 0$ et un pas $\alpha_k \in \mathbb{R}$, $\alpha_k > 0$. On dira que la fonction f diminue suffisamment en $x_k + \alpha_k d_k$ par rapport à x_k si*

$$f(x_k + \alpha_k d_k) \leq f(x_k) + \alpha_k \beta_1 \nabla f(x_k)^T d_k, \tag{4.18}$$

avec $0 < \beta_1 < 1$. La condition (4.18) est la première condition de Wolfe.

Définition 4.4.2 ***(Progrès suffisant : seconde condition de Wolfe [3])*** *Soient $f : \mathbb{R}^n \to \mathbb{R}$ une fonction différentiable, un point $x_k \in \mathbb{R}^n$, un direction de descente $d_k \in \mathbb{R}^n$ telle que $\nabla f(x_k)^T d_k < 0$ et un pas $\alpha_k \in \mathbb{R}$, $\alpha_k > 0$. On dira que le point $x_k + \alpha_k d_k$ apporte un progrès suffisant par rapport à x_k si*

$$\nabla f(x_k + \alpha_k d_k)^T d_k \geq \beta_2 \nabla f(x_k)^T d_k, \tag{4.19}$$

avec $\beta_1 < \beta_2 < 1$, où β_1 est la constante de (4.18). La condition (4.19) s'appelle la seconde condition de Wolfe.

La recherche linéaire inexacte permet d'identifier une longueur de pas qui vérifie les conditions de Wolfe (4.18) et (4.19). L'algorithme suivant, extrait de [3], décrit cette méthode de recherche linéaire inexacte.

Algorithme 4.1 (Recherche linéaire inexacte)

Objectif Trouver une longueur de pas α^* telle que les conditions de Wolfe soient vérifiées.

Entrées

- La fonction $f : \mathbb{R}^n \to \mathbb{R}$ continûment différentiable.
- Le gradient de la fonction $\nabla f : \mathbb{R}^n \to \mathbb{R}^n$.
- Un vecteur $x \in \mathbb{R}^n$.
- Une direction de descente d telle que $\nabla f(x)^T d < 0$.
- Une première approximation de la solution $\alpha_0 > 0$.
- Des paramètres β_1 et β_2 tels que $0 < \beta_1 < \beta_2 < 1$.
- Un paramètre $\lambda > 1$.

Sortie Une longueur de pas α^* telle que les conditions de Wolfe, (4.18) et (4.19), soient vérifiées.

Initialisation $i = 0$, $\alpha_l = 0$, $\alpha_r = +\infty$.

Itérations

1) Si α_i vérifie les conditions (4.18) et (4.19), alors $\alpha^* = \alpha_i$. STOP.
2) Si α_i viole la première condition de Wolfe, alors le pas est trop long et nous posons :

$$\begin{aligned} \alpha_r &= \alpha_i, \\ \alpha_{i+1} &= \frac{\alpha_l + \alpha_r}{2}. \end{aligned}$$

3) Si α_i satisfait la première condition de Wolfe mais viole la deuxième, alors le pas est trop court et nous posons :

$$\begin{aligned} \alpha_l &= \alpha_i, \\ \alpha_{i+1} &= \begin{cases} \dfrac{\alpha_l + \alpha_r}{2} & \text{si } \alpha_r < +\infty \\ \lambda \alpha_i & \text{sinon.} \end{cases} \end{aligned}$$

4) $i = i + 1$.

Avec cette recherche linéaire inexacte, nous obtenons l'algorithme de la plus forte pente (issu de [3]) décrit ci-dessous.

Algorithme 4.2 (Méthode de la plus forte pente)

Objectif Trouver (une approximation de) la solution du problème

$$\min_{x \in \mathbb{R}^n} f(x).$$

Entrées

- La fonction $f : \mathbb{R}^n \to \mathbb{R}$ différentiable.
- Le gradient de la fonction $\nabla f : \mathbb{R}^n \to \mathbb{R}^n$.
- Une première approximation de la solution $x_0 \in \mathbb{R}^n$.
- La précision demandée $\epsilon \in \mathbb{R}$, $\epsilon > 0$.

Sortie Une approximation de la solution $x^* \in \mathbb{R}^n$.

Initialisation $k = 0$.

Itérations

1) Poser $d_k = -\nabla f(x_k)$.
2) Déterminer α_k en appliquant l'Algorithme 4.1 avec $\alpha_0 = 1$.
3) Calculer la mise à jour :

$$x_{k+1} = x_k + \alpha_k d_k.$$

4) $k = k + 1$.

Critère d'arrêt Si $|| \nabla f(x_k) || \leq \epsilon$, alors $x^* = x_k$.

4.4.2 Méthode de Newton

Considérons une fonction $F : \mathbb{R}^n \to \mathbb{R}^n$ continûment différentiable et un point $x_k \in \mathbb{R}^n$. Construisons un modèle linéaire de cette fonction en x_k en nous inspirant de la formule (4.10) du Théorème de Taylor :

$$m_k(x) = F(x_k) + \nabla F(x_k)(x - x_k),$$

où $\nabla F(x_k)$ est la matrice Jacobienne de la fonction F. En égalisant cette linéarisation à 0 et en supposons que $\nabla F(x_k)$ est inversible, nous trouvons alors :

$$\begin{aligned} x &= x_k - \nabla F(x_k)^{-1} F(x_k) \\ &\overset{not.}{=} x_{k+1}. \end{aligned} \tag{4.20}$$

Prenons maintenant le cas où $F = \nabla f$, l'expression (4.20) de la mise à jour x_{k+1} devient :

$$x_{k+1} = x_k - \nabla^2 f(x_k)^{-1} \nabla f(x_k). \tag{4.21}$$

Le nouvel itéré x_{k+1} est calculé à partir de l'itéré précédent x_k en appliquant une formule de la forme $x_{k+1} = x_k + d_k$ où la direction de recherche d_k, appelée *direction de Newton*, est de la forme :

$$d_k \quad = \quad -\nabla^2 f(x_k)^{-1} \nabla f(x_k). \tag{4.22}$$

Nous retrouvons la direction de recherche introduite auparavant par l'expression (4.12) avec $B_k = \nabla^2 f(x_k)$. Cette direction de Newton est une direction de descente en x pour f lorsque $\nabla^2 f(x)$ est définie positive. De plus, nous pouvons remarquer que la longueur du pas est égale à 1 dans cette méthode.

Comme l'énoncent J. Nocedal et S. J. Wright [28], pour tout x dans le voisinage d'une solution x^* telle que $\nabla^2 f(x^*)$ est définie positive, la matrice Hessienne $\nabla^2 f(x)$ sera également définie positive. La méthode de Newton est donc bien définie dans cette région et converge quadratiquement, à condition que la longueur du pas soit toujours 1. La *méthode de Newton*, décrite par l'Algorithme 4.3 extrait de [3], est donc une méthode locale. Afin d'obtenir une convergence globale de la méthode de Newton, nous pouvons la combiner avec une recherche linéaire.

Algorithme 4.3 (Méthode de Newton)

Objectif Trouver (une approximation de) la solution du problème

$$\min_{x \in \mathbb{R}^n} f(x).$$

Entrées

- La fonction $f : \mathbb{R}^n \to \mathbb{R}$ différentiable.
- Le gradient de la fonction $\nabla f : \mathbb{R}^n \to \mathbb{R}^n$.
- Une première approximation de la solution $x_0 \in \mathbb{R}^n$.
- La précision demandée $\epsilon \in \mathbb{R}$, $\epsilon > 0$.

Sortie Une approximation de la solution $x^* \in \mathbb{R}^n$.

Initialisation $k = 0$.

Itérations

1) Calculer la direction de Newton, d_k, solution de :

$$\nabla^2 f(x_k) d_k \quad = \quad -\nabla f(x_k).$$

2) Calculer la mise à jour :

$$x_{k+1} = x_k + d_k.$$

3) $k = k + 1$.

Critère d'arrêt Si $\| \nabla f(x_k) \| < \epsilon$, alors $x^* = x_k$.

4.4.3 Problème des moindres carrés

Dans le cadre des *problèmes de moindres carrés*, la fonction objective f à une forme particulière. Ces problèmes d'optimisation sont de la forme suivante :

$$\begin{aligned} \min_{x\in\mathbb{R}^n} f(x) &= \min_{x\in\mathbb{R}^n} \frac{1}{2}\sum_{i=1}^{m} g_i(x)^2, \qquad (4.23)\\ &= \min_{x\in\mathbb{R}^n} \frac{1}{2}\, g(x)^T g(x),\\ &= \min_{x\in\mathbb{R}^n} \frac{1}{2}\, \|g(x)\|^2, \qquad (4.24) \end{aligned}$$

où $m \geq n$ et chaque g_i est une fonction continûment différentiable de $\mathbb{R}^n$ dans $\mathbb{R}$. Afin d'obtenir la nouvelle écriture (4.24), nous définissons un vecteur $g : \mathbb{R}^n \to \mathbb{R}^m$ contenant les composantes g_i :

$$g(x) = (g_1(x), g_2(x), \ldots, g_m(x))^T.$$

Dans le but d'appliquer la méthode de Newton locale au problème des moindres carrés, calculons le gradient (4.25) et la matrice Hessienne (4.26) de ce problème :

$$\begin{aligned} \nabla f(x) &= \sum_{i=1}^{m} g_i(x)\ \nabla g_i(x)\\ &= \nabla g(x)^T g(x), \qquad (4.25) \end{aligned}$$

et

$$\begin{aligned} \nabla^2 f(x) &= \sum_{i=1}^{m} \left(\nabla g_i(x) \nabla g_i(x)^T + g_i(x)\ \nabla^2 g_i(x)\right)\\ &= \nabla g(x)^T \nabla g(x) + \sum_{i=1}^{m} g_i(x)\ \nabla^2 g_i(x), \qquad (4.26) \end{aligned}$$

où $\nabla g(x) \in \mathbb{R}^{m\times n}$ est la matrice Jacobienne de la fonction g et $\nabla g_i(x) \in \mathbb{R}^{1\times n}$ est le gradient de g_i. Le deuxième terme dans l'expression de la matrice Hessienne est, en générale, très coûteux à calculer en pratique. En effet, le calcul de ce terme implique les dérivées secondes des fonctions g_i. Pour contourner ce problème, nous approximons la matrice Hessienne par le premier terme :

$$\nabla^2 f(x) \approx \nabla g(x)^T \nabla g(x).$$

En utilisant ces expressions du gradient et de la matrice Hessienne, nous pouvons générer la direction de recherche d_k en résolvant les équations standards de Newton $\nabla^2 f(x_k)d = -\nabla f(x_k)$ qui deviennent :

$$\nabla g(x_k)^T \nabla g(x_k) d = -\nabla g(x_k)^T g(x_k).$$

La méthode obtenue, appelée la *méthode de Gauss-Newton*, peut être vue comme une modification de la méthode de Newton. L'algorithme 4.2, extrait de l'ouvrage de M. Bierlaire [3], décrit cette méthode.

Algorithme 4.4 (Méthode de Gauss-Newton)

Objectif : Trouver la solution du problème aux moindres carrés

$$\min_{x\in\mathbb{R}^n} f(x) \quad = \quad \min_{x\in\mathbb{R}^n} \frac{1}{2}\, g(x)^T g(x).$$

Entrées :

- La fonction différentiable $g : \mathbb{R}^n \to \mathbb{R}^m$.
- La matrice Jacobienne $\nabla g : \mathbb{R}^n \to \mathbb{R}^{m\times n}$.
- Une première approximation de la solution $x_0 \in \mathbb{R}^n$.
- La précision demandée $\epsilon \in \mathbb{R}$, $\epsilon > 0$.

Sortie : Une approximation de la solution $x^* \in \mathbb{R}^n$.

Initialisation : $k = 0$.

Itérations :

1) Calculer d_{k+1} solution de :

$$\nabla g(x_k)^T \nabla g(x_k) d_{k+1} \quad = \quad -\nabla g(x_k)^T\, g(x_k). \tag{4.27}$$

2) Calculer la mise à jour :

$$x_{k+1} \quad = \quad x_k + d_{k+1}. \tag{4.28}$$

3) $k = k + 1$.

Critère d'arrêt : Si $\left\|\nabla g(x_k)^T g(x_k)\right\| \leqslant \epsilon$, alors $x^* = x_k$.

Lorsque chaque fonction g_i est linéaire, la matrice Jacobienne $\nabla g(x)$ est constante. Cette matrice est de dimension $m \times n$ et nous la notons C. Nous obtenons alors le *problème aux moindres carrés linéaire* de la forme :

$$\min_{x\in\mathbb{R}^n} f(x) \quad = \quad \min_{x\in\mathbb{R}^n} \frac{1}{2}\, \|Cx + b\|_2^2\,, \tag{4.29}$$

où $b = g(0) \in \mathbb{R}^n$. En calculant le gradient et la matrice Hessienne de ce nouveau problème, nous obtenons :

$$\nabla f(x) \quad = \quad C^T(Cx + b), \tag{4.30}$$
$$\nabla^2 f(x) \quad = \quad C^T C, \tag{4.31}$$

puisque chaque $\nabla^2 g_i(x)$ est égal à 0 dans l'expression (4.26). De cette manière, l'équation de Gauss-Newton (4.27) devient

$$C^T C x_{k+1} \quad = \quad -C^T b. \tag{4.32}$$

La dernière égalité (4.32) décrit un système d'équations, appelé le système des *équations normales* du problème des moindres carrés linéaire.

Chapitre 5

Premiers pas dans la librairie ITK

Les deux premières sections de ce chapitre passent en revue et détaillent les outils informatiques utilisés par le CHU de Mont-Godinne et donc pour la réalisation de ce mémoire. Comme notre objectif est d'étudier les méthodes d'optimisation implémentées dans les algorithmes de recalage d'images qui se trouvent dans la *librairie ITK*, cette bibliothèque de traitement d'images est donc le premier outil présenté et celui qui sera le plus utilisé dans la suite. Nous exposons également le logiciel de traitement et de visualisation d'images, *MeVisLab*. Dans notre cas, nous l'utilisons uniquement pour la visualisation car ce logiciel propose plusieurs "démos" dont un recalage d'images réalisé avec la librairie ITK. Enfin, nous introduisons le logiciel *CMake* qui facilite la compilation et le développement des algorithmes que nous pouvons trouver dans la librairie ITK. Dans le cadre de ce mémoire, ce logiciel a juste été utilisé pour l'installation de cette librairie.

Dans la troisième section de ce chapitre, nous parlons des premières pistes que nous avons suivies concernant l'optimisation dans le cadre du recalage médical. Grâce à la littérature concernant le fonctionnement de la librairie ITK et au logiciel MeVisLab, nous avons été attirés par certaines *méthodes d'optimisation* de la librairie ITK. Nous faisons donc le lien entre l'implémentation de ces méthodes et la théorie présentée dans le Chapitre 4.

5.1 Logiciels et bibliothèque d'imagerie

5.1.1 Traitement d'images

La boîte à outils de segmentation et de recalage ITK (*Insight Segmentation and Registration Toolkit* [21]) est un framework C++ open-source (gratuit) et multi-plateformes (compatible avec Linux, Windows, etc). ITK, dont le logo est illustré à la Figure 5.1, fournit aux développeurs un ensemble d'outils logiciels pour l'analyse d'images. Cette bibliothèque contient des algorithmes de pointe pour le *traitement*, le *recalage* et la *segmentation* de données multidimensionnelles.

ITK a initialement été développé grâce au soutien de l'Institut américain de la santé en 1999. Cette librairie est le produit d'un consortium financé par le NIH (National Institute of Health) des Etats Unis, composé de trois industries (Kitware, GE Corporate R&D et Insightful) et de trois universités américaines (Université de Caroline du Nord, Université de Utah et Université de Pennsylvanie). Le but premier était de faciliter la manipulation des données four-

FIGURE 5.1 – Logo de la librairie ITK [21]

nies par le projet "Visible Human Project". ITK a été développé, d'une manière plus générale, pour supporter des applications d'imageries médicales et pour réduire les coûts en temps du développement de telles applications.

ITK est donc un ensemble de codes de recalage et de segmentation d'images. La segmentation est un procédé permettant d'identifier et de classifier l'information sur l'image, nous n'utilisons pas ITK pour cela. Le recalage, la partie d'ITK utilisée dans ce mémoire, sert à aligner ou mettre en évidence des correspondances entre les informations dans l'image, comme nous l'avons présenté dans les deuxième et troisième chapitres. Cette librairie ne prend pas en charge les aspects de visualisation et d'interface graphique. Ces fonctionnalités sont laissées à d'autres produits comme QT [15] pour l'interface graphique et VTK [20] pour la visualisation.

Comme ITK est une librairie open-source et multi-plateformes, nous pouvons la télécharger gratuitement sur le site ITK [21]. Nous obtenons alors un fichier contenant différents dossiers qui, eux-mêmes, contiennent des codes C++ de segmentation, de recalage, d'optimisation et bien d'autres choses qui ne nous intéressent pas dans notre cas.

5.1.2 Visualisation et traitement d'images

MeVisLab [26] est un logiciel de *traitement* et de *visualisation* d'images médicales qui constitue un cadre puissant pour le développement d'algorithmes de traitement d'images, ainsi que des méthodes de visualisation et d'interaction, avec un accent particulier sur l'imagerie médicale.

Outre les modules de visualisation et de traitement d'images de base, MeVisLab, dont le logo est illustré à la Figure 5.2, inclut des algorithmes avancés d'imagerie médicale pour la segmentation, le recalage et l'analyse d'images morphologiques et fonctionnelles.

FIGURE 5.2 – Logo du logiciel MeVisLab [26]

L'outil de développement MeVisLab a commencé en 1993 avec le logiciel ILAB1 de l'Institut CEVIS, écrit en C++. Il peut se connecter de manière interactive à des algorithmes pour former des réseaux de traitement d'images. En 1995, le centre de recherche MeVis, nouvellement créé, a repris le développement et a publié ILAB2 et ILAB3 [56].

5.2 L'outils de développement CMake

CMake [8], abréviation de "Cross-platform Make", est un *système de compilation*, libre, open-source (gratuit) et multi-plateformes (compatible avec Linux, Windows, etc). CMake, dont le logo est représenté à la Figure 5.3, est une famille d'outils conçus pour construire, développer et tester des logiciels. Il est utilisé pour contrôler le processus de compilation d'une application et pour la rendre indépendante de l'environnement du compilateur. CMake a été créé par Kitware afin de répondre au besoin d'un constructeur puissant et multi-plateformes pour des projets open-source tels que ITK et VTK.

FIGURE 5.3 – Logo du logiciel CMake [8]

Avant d'utiliser CMake, il faut écrire, selon une syntaxe particulière, un fichier source de configuration, appelé "CMakeLists.txt". Celui-ci comporte tous les paramètres de l'application et la démarche à suivre pour la construire. C'est notamment dans ce fichier que l'on spécifie les liens vers les librairies nécessaires à la compilation d'un programme. Une fois qu'il est créé, le rôle du logiciel CMake est de générer automatiquement plusieurs fichiers dont la procédure de construction peut être résumée par la Figure 5.4. Cette figure montre qu'à partir du "CMakeLists.txt" qui se trouve dans la librairie, le logiciel CMake génère plusieurs fichiers : "CMakeCache.txt", "CMakeFiles" et "Makefile".

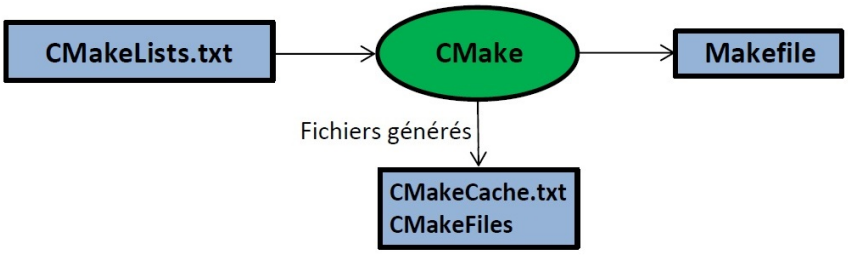


FIGURE 5.4 – Procédure de construction de CMake [16]

Le fichier "CMakeCache.txt" définit un ensemble de variables de configuration et d'informations sur le système. Le répertoire "CMakeFiles" contient les fichiers intermédiaires de compilation, des fichiers temporaires et autres fichiers de configuration qui ne regardent que CMake. Le seul fichier qui nous intéresse est le "Makefile" qui est tout à fait adapté à l'environnement de compilation du programmeur.

Lorsque nous utilisons des librairies telles que ITK ou VTK, CMake est un outils indispensable car les "Makefiles" sont souvent de grande taille. Dans le cadre de ce mémoire, CMake a juste été utilisé afin d'installer la librairie ITK. Cependant, nous laissons l'explication de la démarche à suivre pour compiler des codes au cas où une personne voudrait poursuivre ce sujet

et aller plus loin dans l'optimisation des codes. De plus, cette explication permet également de comprendre à quoi servent les "CMakeLists.txt" que nous trouvons à plusieurs endroits dans la librairie ITK.

5.3 L'optimisation dans le recalage

Le but de ce mémoire est d'étudier l'optimisation des algorithmes de recalage. Nous avons donc été attirés par le dossier "Optimizer" (`\ITK\Modules\Numerics\Optimizers`) qui se trouve dans la librairie ITK. Parmi les codes d'optimisation, nous retrouvons plusieurs méthodes d'optimisation comme la méthode de Broyden-Fletcher-Goldfarb-Shanno (BFGS) [28], la méthode de descente du gradient dont la théorie est présentée dans le Chapitre 4 sous le nom de la méthode de plus forte pente et la méthode de Levenberg-Marquardt [28]. L'ensemble des méthodes sont reprises à la Figure 5.5 où nous avons mis en évidence, en les encadrant en rouge, les méthodes citées ci-dessus.

Dans cette librairie, il existe également des implémentations de plusieurs exemples de recalage d'images dans le dossier "Examples" (`\ITK\Examples\Registration`). La Table 5.1 reprend le nom des fichiers C++ des différents exemples, la transformation appliquée et la méthode d'optimisation permettant de minimiser la distance entre les deux images à recaler.

File Name	**Transform**	**Optimizer**
ImageRegistration1.cxx	Translation	Regular Step Gradient Descent
ImageRegistration2.cxx	Translation	Gradient Descent
ImageRegistration3.cxx	Translation	Gradient Descent
ImageRegistration4.cxx	Translation	Gradient Descent
ImageRegistration5.cxx	CenteredRigid2D	Gradient Descent
ImageRegistration6.cxx	CenteredRigid2D	Gradient Descent
ImageRegistration7.cxx	CenteredSimilarity2D	Gradient Descent
ImageRegistration8.cxx	VersorRigid3D	Gradient Descent
ImageRegistration9.cxx	Affine	Gradient Descent
ImageRegistration10.cxx	Translation	Gradient Descent
ImageRegistration11.cxx	Translation	Gradient Descent
ImageRegistration12.cxx	CenteredRigid2D	Gradient Descent
ImageRegistration13.cxx	CenteredRigid2D	Gradient Descent
ImageRegistration14.cxx	CenteredRigid2D	Gradient Descent
ImageRegistration15.cxx	Translation	Gradient Descent
ImageRegistration16.cxx	Translation	Gradient Descent
ImageRegistration17.cxx	Translation	Gradient Descent
ImageRegistration18.cxx	Translation	Gradient Descent
ImageRegistration19.cxx	Affine	Gradient Descent
ImageRegistration20.cxx	Affine	Regular Step Gradient Descent

TABLE 5.1 – Exemples d'algorithmes de recalage - Extrait de "Public Wiki" [31]

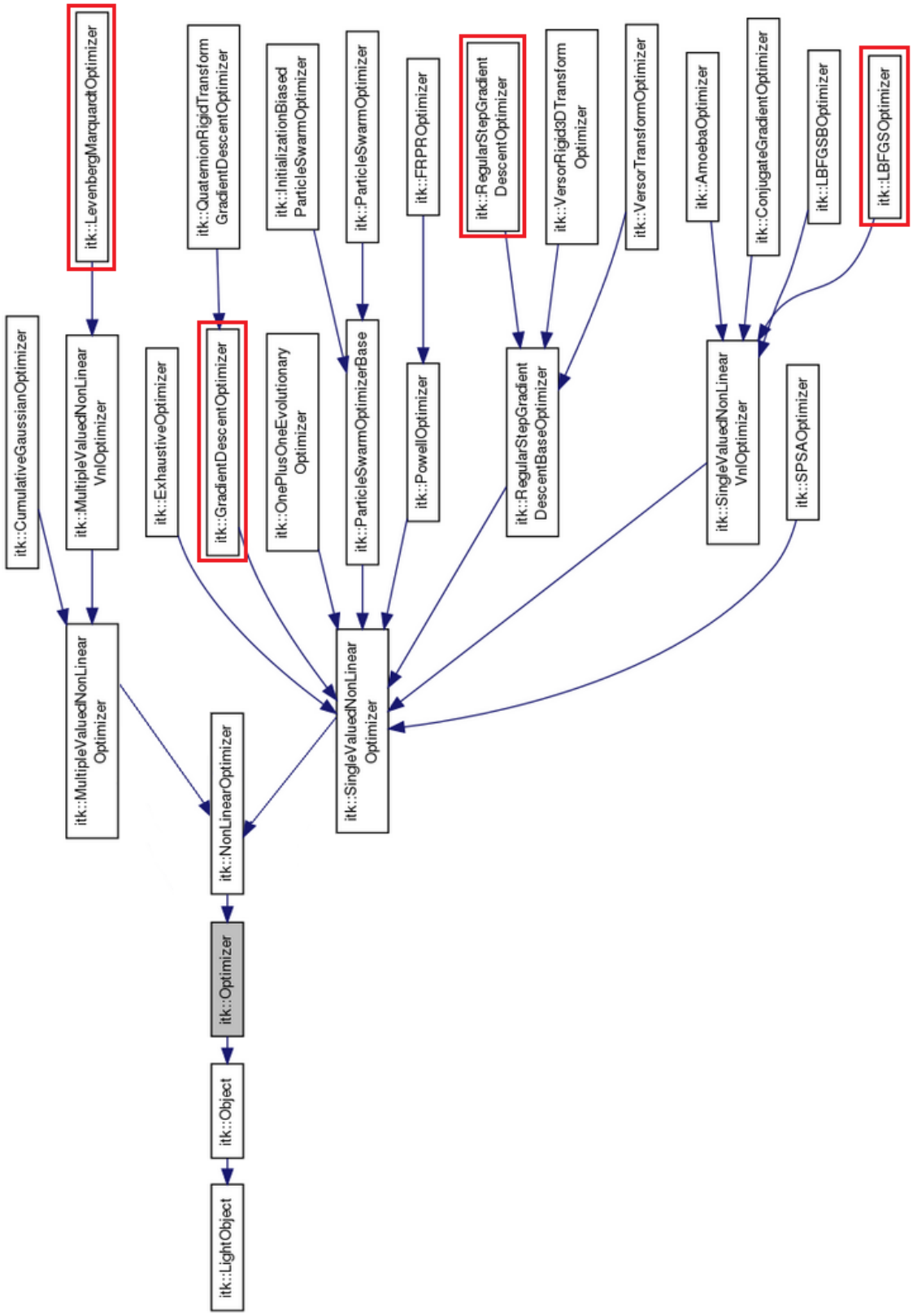


FIGURE 5.5 – Méthodes d'otpimisation de la librairie ITK [12]

Comme le montre la deuxième colonne de la Table 5.1, il existe différentes transformations possibles. Pour certains exemples, ce sont des transformations rigides comme la translation, pour d'autres exemples, ce sont des transformations non rigides comme les transformations affines. Ces différentes transformations ont été présentées précédemment dans la Section 3.4 du Chapitre 3. La troisième colonne de la Table 5.1 contient le nom des méthodes d'optimisation appliquées. Les exemples utilisent deux variantes de la *méthode de descente du gradient*.

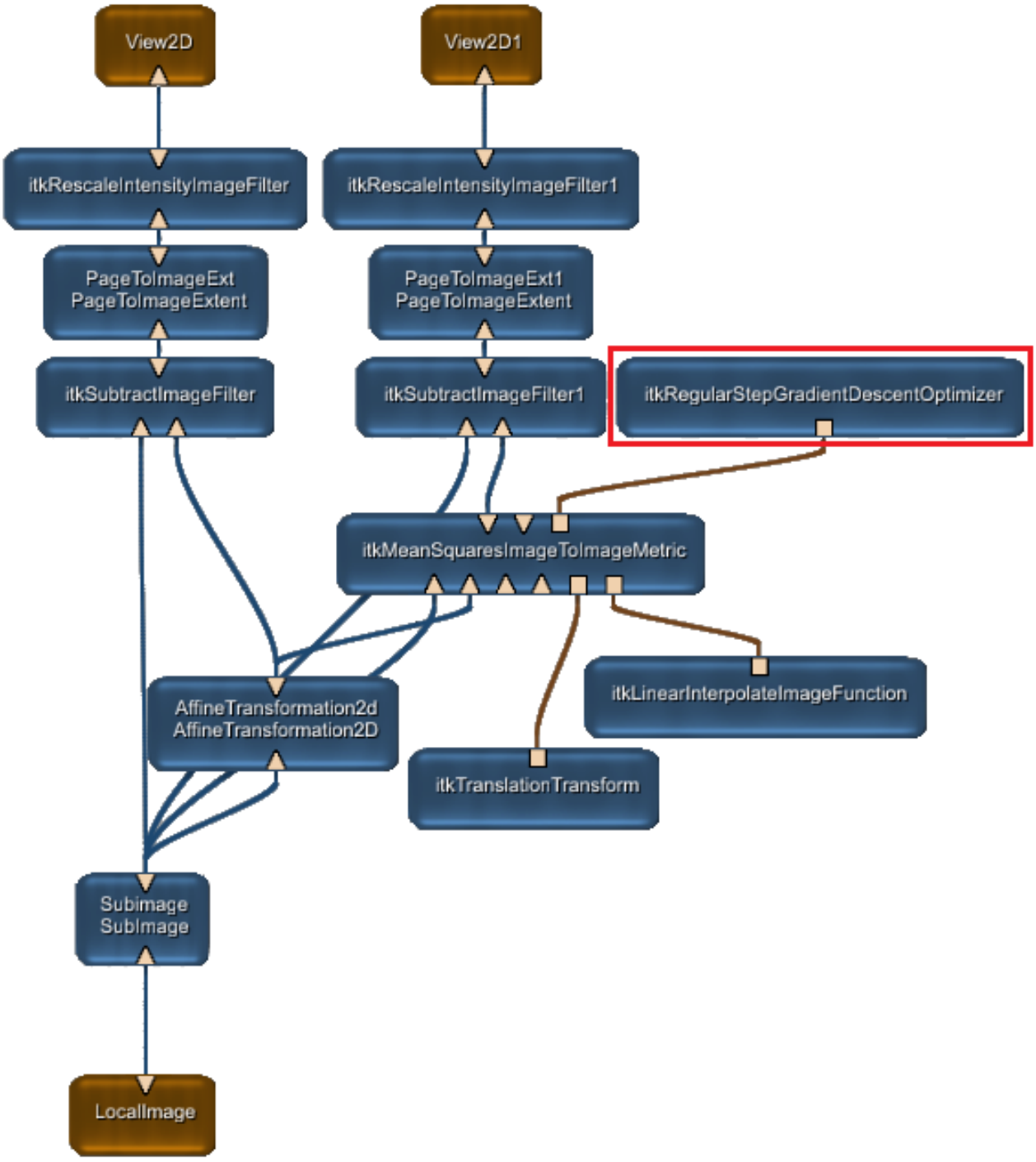


FIGURE 5.6 – "Arbre" d'un exemple de recalage d'images fournit par MevisLab [26]

Comme nous l'avons expliqué au début de ce chapitre, il existe également le logiciel de visualisation MeVisLab qui propose plusieurs "démos" dont un exemple de recalage d'images réalisé grâce à la librairie ITK. Ce logiciel fournit des "arbres" ou des "réseaux" comme celui de la Figure 5.6. Cet arbre indique le cheminement permettant de recaler deux images à deux dimensions. Nous pouvons voir que la méthode illustrée sur cette figure part de deux images, nommées "View2D" et "View2D1". Ensuite, la méthode fait appel à de nombreux codes afin de fournir l'image recalée, appelée "LocalImage". Comme le montre le cadre rouge, cet exemple de recalage utilise, de nouveau, une variante de la méthode de descente du gradient comme méthode d'optimisation. Même si le dossier "Optimizer" de la librairie ITK contient de nombreuses autres méthodes d'optimisation (Broyden-Fletcher-Goldfarb-Shanno, Levenberg-Marquardt, etc.), nous nous concentrons sur les méthodes les plus utilisées dans le cadre du recalage d'images, à savoir les deux variantes de la méthode de descente du gradient.

Dans le Chapitre 4 (Section 4.4.1), nous avons présenté la théorie concernant cette méthode de descente du gradient, appelée la méthode de plus forte pente. Nous pouvons donc maintenant faire le lien entre l'implémentation de ces deux variantes de la méthode et la théorie. Ces deux algorithmes d'optimisation, "Regular Step Gradient Descent" et "Gradient Descent", sont implémentés en C++ et se trouvent en annexe en version "nettoyés". En effet, nous n'avons pas mis les algorithmes comme nous les avons trouvés dans la librairie, nous avons supprimé les lignes qui ne permettaient pas de comprendre les méthodes appliquées. Après plusieurs essais, nous avons pu retranscrire ces deux méthodes en pseudo-code, ce sont les Algorithmes 5.1 et 5.2 ci-après.

Algorithme 5.1 (Méthode de descente du gradient avec un pas régulier)

Objectif Trouver (une approximation de) la solution du problème

$$\min_{x \in \mathbb{R}^n} f(x).$$

Entrées :

- La fonction $f : \mathbb{R}^n \to \mathbb{R}$ différentiable.
- Le gradient de la fonction $\nabla f : \mathbb{R}^n \to \mathbb{R}^n$.
- La dimension de l'espace considéré n.
- La position initiale $x_0 \in \mathbb{R}^n$.
- La précision demandée $\epsilon \in \mathbb{R}$, $\epsilon > 0$.

Initialisations :

- La longueur maximale du pas $\beta\_max = 1$.
- La longueur minimale du pas $\beta\_min = 0.001$.
- La précision demandée $\epsilon = 0.0001$.
- Le nombre maximum d'itérations $Iter = 100$.
- L'itération courante $k = 0$.
- La longueur du pas $\beta = \beta\_max$.

- Le facteur de relaxation $Factor = 0.5$.
- Le produit scalaire $PS = 0$.

Itérations :

1) Si $k \geqslant Iter$, alors STOP.
2) Calculer la direction : $d_k = -\nabla f(x_k)$.
3) Si $\|\nabla f(x_k)\|^2 < \epsilon$, alors STOP.
4) Si $k = 0$, alors $PS = 0$, sinon calculer le produit scalaire :

$$PS \quad = \quad \nabla f(x_k)^T \nabla f(x_{k-1}).$$

5) S'il y a un changement de direction ($PS < 0$), alors $\beta = \beta.Factor$.
6) Si $\beta < \beta\_min$, alors STOP.
8) Calculer la longueur du pas :

$$\alpha_k \quad = \quad \frac{\beta}{\|\nabla f(x_k)\|}.$$

9) Calculer la mise à jour :

$$x_{k+1} \quad = \quad x_k + \alpha_k d_k.$$

10) $k = k + 1$.

Aux pas 2), 8), 9) et 10), nous retrouvons clairement les étapes de l'algorithme de la plus forte pente. La seule différence réside dans le calcul de la longueur du pas, α_k. Dans la théorie, il faut appliquer une recherche linéaire afin de trouver cette valeur. Dans la librairie ITK, nous regardons s'il y a un changement de direction entre le gradient courant et le gradient précédent et si c'est le cas, nous divisons la longueur du pas, β, par deux. Pour ce faire, nous multiplions β par "$Factor$" qui est égal à 0.5. Ensuite, la longueur du pas, α_k, devient la longueur du pas normalisée par la norme du gradient de la fonction.

Comme cette méthode n'utilise pas de recherche linéaire pour définir la longueur du pas, c'est une version très simplifiée de la méthode de la plus forte pente. Cependant, avec des problèmes de grandes dimensions, comme nous pouvons trouver en recalage d'images, il est assez coûteux de faire une recherche linéaire. Passons maintenant à la deuxième variante de la méthode avec l'Algorithme 5.2.

Algorithme 5.2 (Méthode de descente du gradient)

Entrées :

- La fonction $f : \mathbb{R}^n \to \mathbb{R}$ différentiable.
- Le gradient de la fonction $\nabla f : \mathbb{R}^n \to \mathbb{R}^n$.
- La dimension de l'espace considéré n.
- La position initiale $x_0 \in \mathbb{R}^n$.

Initialisations :

- Le nombre maximum d'itérations $Iter = 100$.
- L'itération courante $k = 0$.

Itérations :

1) Si $k \geqslant Iter$, alors STOP.
2) Calculer la direction : $d_k = -\nabla f(x_k)$.
3) Calculer la mise à jour :

$$x_{k+1} = x_k + d_k.$$

4) $k = k + 1$.

L'implémentation ci-dessus est encore plus basique que la précédente. Dans ce cas, il n'y a aucun calcul de la longueur du pas, nous pouvons supposer qu'elle est fixée à 1. La direction de descente est égale à l'opposé du gradient, comme pour la méthode de la plus forte pente. Ensuite, la mise à jour est égale à l'itéré courant augmenté de cette direction de descente.

5.4 Conclusion

Dans ce chapitre, nous avons présenté la librairie ITK qui est la boîte à outils de segmentation et de recalage utilisée par le CHU de Mont-Godinne. Nous avons également fait nos premiers pas dans cette librairie en trouvant plusieurs méthodes d'optimisation implémentées. Enfin, nous nous sommes concentrés sur une méthode en particulier et nous avons fait le lien entre la théorie et les codes.

A ce stade du travail, nous nous intéressons plus particulièrement à la méthode de recalage d'images, nommée la *méthode des démons*, car c'est la méthode de recalage utilisée par le CHU. Malheureusement, il n'existe pas de méthode "Optimizer" utilisée en tant que telle dans l'algorithme des démons. En effet, cet algorithme n'utilise pas les méthodes d'optimisation citées et présentées auparavant. Dans ce cas, l'optimisation est directement gérée dans la boucle générale de l'algorithme des démons comme nous allons le voir dans le chapitre suivant.

Chapitre 6

Algorithme des démons

Comme nous l'avons expliqué précédemment, le recalage d'images est le processus d'alignement des images qui permet aux caractéristiques de ces images d'être mises en correspondance. Le recalage aligne les images en appliquant des transformations à l'une des images pour qu'elle corresponde à l'autre. Rappelons également que le recalage peut être divisé en recalage rigide et en recalage non rigide, aussi appelé recalage déformable.

Dans le domaine médical, différentes méthodes de recalage déformable sont utilisées. Parmi ces méthodes, les développeurs utilisent principalement la *technique des démons* proposée par J.-P. Thirion [42] en raison de son efficacité et de sa simplicité. Dans ce chapitre, nous expliquons donc le principe de l'algorithme des démons, ensuite nous détaillons l'algorithme ainsi que tous les outils mathématiques nécessaires pour le développer. Nous terminons en faisant le lien entre l'algorithme et son implémentation dans la librairie ITK.

6.1 Principe de l'algorithme des démons

Selon J.-P. Thirion [42], le recalage d'images est comparable à un *processus de diffusion* d'une image vers une autre. Les objets apparaissant dans l'image fixe retrouvent leur vis-à-vis dans l'image mouvante. De plus, pour passer d'une image à l'autre, ces objets doivent subir une diffusion ou une déformation.

Pour comprendre l'idée principale de l'algorithme des démons, il faut considérer le contour de l'objet de l'image fixe comme une membrane semi-perméable et considérer l'image mouvante comme une grille déformable. Chaque sommet de cette grille peut être étiqueté de deux manières :

- "à l'intérieur" signifie que sa position relative dans l'image fixe se situe à l'intérieur du contour,
- "à l'extérieur" signifie que sa position relative dans l'image fixe se situe à l'extérieur du contour.

De plus, sur le contour de l'objet de l'image fixe, il y a des *effecteurs* ou encore des *"démons"* qui permettent la diffusion de l'image déformée au travers du contour de l'image statique. Ce processus est illustré à la Figure 6.1 où une image déformée, portée par une grille déformable, est diffusée au travers des contours de l'objet de l'image fixe sous l'action des démons.

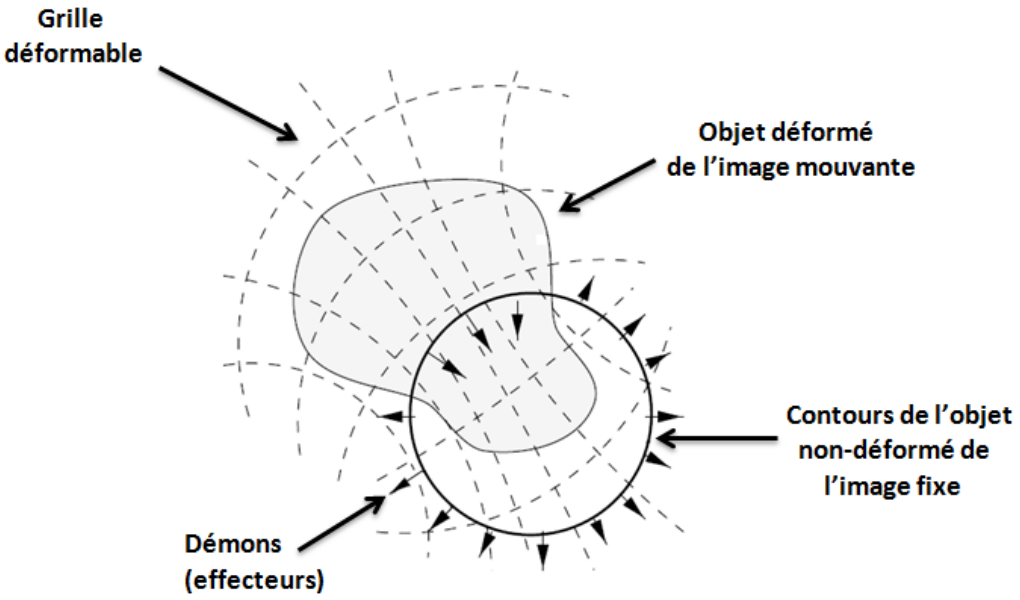


FIGURE 6.1 – Modèle de diffusion de la méthode des démons [42]

Selon J.-P. Thirion, un démon est un effecteur ou un opérateur situé en un point P du contour d'un objet O de l'image fixe. Le rôle du démon est de déformer la grille pour pousser l'objet déformé de l'image mouvante à l'intérieur de O si le sommet correspondant est classé "à l'intérieur" et à l'extérieur de O si le sommet correspondant est classé "à l'extérieur". Chaque démon est donc associé à un vecteur perpendiculaire au contour qui traduit l'action du démon.

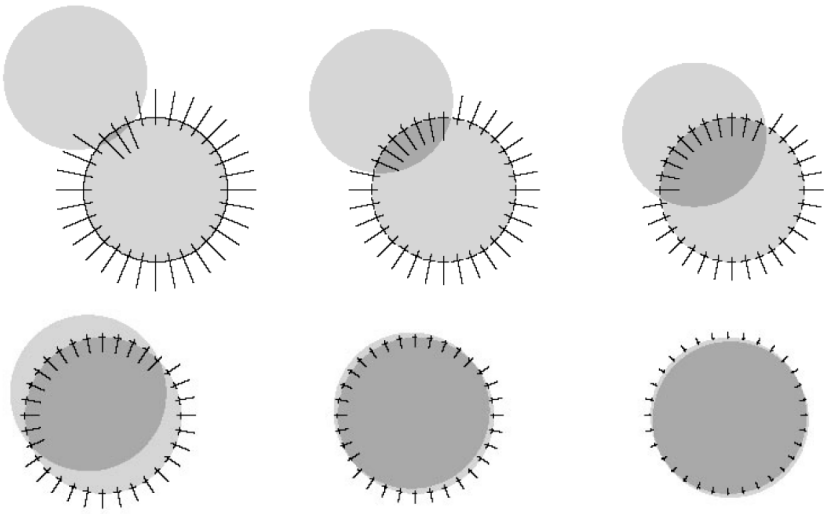


FIGURE 6.2 – Quelques itérations d'un modèle de diffusion [42]

Afin d'illustrer ce principe, nous considérons le cas simple de deux images représentant le même disque et nous nous limitons à des transformations rigides (Figure 6.2). Un ensemble de démons est placé sur le contour du disque de l'image fixe. La force des démons est orientée de

l'intérieur vers l'extérieur du disque lorsque le point du modèle correspondant est étiqueté "à l'extérieur", et dans l'autre sens si l'étiquette est "à l'intérieur". Comme le montre la Figure 6.2, cette méthode est itérative. A chaque itération, le mouvement engendré par toutes les forces est appliqué au modèle. La grandeur des forces induites par les démons diminue à chaque itération afin d'atteindre une convergence. Lorsque les deux disques ne se chevauchent pas au départ comme sur la Figure 6.3, la méthode des démons ne s'applique pas.

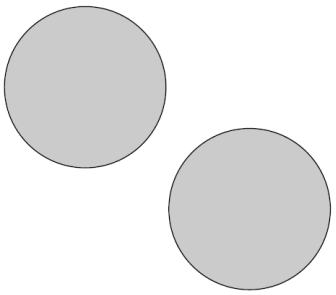


FIGURE 6.3 – Cas problématique d'un modèle de diffusion [42]

6.2 Algorithme des démons

Maintenant que nous connaissons le fonctionnement de cet algorithme, nous pouvons aller plus dans les détails afin d'arriver à la formulation de l'algorithme en lui-même. Pour ce faire, nous nous basons sur l'article de T. Vercauteren [46].

Considérons une image mouvante $M : \Omega \subseteq \mathbb{R}^d \longrightarrow \mathbb{R}$ et une image fixe $F : \Omega \subseteq \mathbb{R}^d \longrightarrow \mathbb{R}$, où Ω est l'ensemble des pixels p. Le recalage d'images est traité comme un problème d'optimisation qui a pour objectif de trouver le déplacement de chaque pixel afin d'obtenir un alignement correct des deux images. La transformation $s : \Omega \longrightarrow \Omega : p \mapsto s(p)$ modélise le déplacement des pixels de l'image mouvante vers l'image fixe.

Comme nous l'avons expliqué dans le Chapitre 3, le critère de similarité $Sim(\ ,\ )$ mesure la ressemblance des deux images. Dans notre cas, nous considérons l'*erreur quadratique moyenne* qui est le critère le plus utilisé dans le recalage fondé sur l'intensité :

$$Sim(F, M \circ s) = \frac{1}{2} \|F - M \circ s\|^2 \tag{6.1}$$

$$= \frac{1}{2} \sum_{p \in \Omega} (F(p) - (M \circ s)(p))^2 , \tag{6.2}$$

où $M \circ s$ est l'image mouvante ayant subi le déplacement s. Une simple optimisation de (6.1) conduit à un problème mal posé avec des solutions instables et non lisses. Pour éviter ceci, un terme de régularisation, $Reg(s)$, est introduit pour obtenir l'expression de l'énergie :

$$E(s) = Sim(F, M \circ s) + \frac{1}{\sigma_T^2} Reg(s), \tag{6.3}$$

où σ_T contrôle la quantité de régularisation dont nous avons besoin. Cette énergie offre un problème bien posé, mais le mélange de la similarité et des modalités de régularisation conduit, en général, à des étapes de calcul d'optimisation intensif.

Afin de rendre les calculs d'optimisation plus simples, nous introduisons une variable cachée dans le processus de recalage, cette variable est appelée la *correspondance*. Cette correspondance est un champ de vecteurs $c = s+u$, où u est le vecteur de mise jour. Nous obtenons ainsi l'énergie globale suivante :

$$E(c,s) = Sim(F, M \circ c) + \sigma_i^2 dist(s,c)^2 + \frac{1}{\sigma_T^2} Reg(s) \quad (6.4)$$

$$= Sim(F, M \circ c) + \sigma_i^2 \|c - s\|^2 + \frac{1}{\sigma_T^2} Reg(s), \quad (6.5)$$

ou encore,

$$E_s^{corr}(u) = Sim(F, M \circ (s+u)) + \sigma_i^2 \|u\|^2 + \frac{1}{\sigma_T^2} Reg(s), \quad (6.6)$$

où $dist(s,c)$ est la distance entre s et c et σ_i^2 représente le bruit sur l'intensité de l'image en chaque pixel.

L'algorithme des démons de J.-P. Thirion est un processus itératif à deux étapes : 1) le *calcul d'un vecteur de déplacement* qui correspond à la dérivée variationnelle de la mesure de similarité, et 2) une *régularisation* par un lissage gaussien de ce vecteur de déplacement. Nous obtenons ainsi l'algorithme des démons, extrait de l'article de J. D. Owens [29] :

Algorithme 6.1 (Méthode des démons)

Calculer $\nabla_p F$ pour chaque pixel p.

Initialisation de u à 0.

REPETER

POUR chaque pixel p :

1) Calculer la valeur de l'intensité dans l'image mouvante : $M \circ s(p)$.
2) Calculer le vecteur de déplacement additionnel $u(p)$, c'est-à-dire la mise à jour, en minimisant

$$E_s^{corr}(u) = \frac{1}{2} \|F - M \circ (s+u)\|^2 + \frac{\sigma_i^2}{2} \|u\|^2 \quad (6.7)$$

par rapport à u.

3) Calculer le vecteur de déplacement total :

$$u_{total}(p) = u_{total}(p) + u(p).$$

FIN.

Régularisation de u_{total} en appliquant un lissage gaussien.

JUSQU'A M et F convergent $\left(M \xrightarrow{Converge} F\right)$.

De manière générale, le lissage est une technique de réduction du bruit qui permet de préserver une haute qualité de l'image. La technique est spécialement conçue pour supprimer les pointes de bruit, c'est-à-dire les pixels isolés d'une intensité de pixel exceptionnellement haute ou basse. Le lissage gaussien est un cas particulier qui utilise, pour ce faire, la loi de probabilité de Gauss [54].

Afin de trouver l'expression de la mise à jour, nous devons minimiser l'énergie de correspondance définie à l'étape 2 de l'algorithme des démons. Pour ce faire, il faut d'abord transformer le problème de minimisation en un *problème aux moindres carrés linéaire*. Ensuite, nous devons trouver le minimum de ce problème en résolvant un système linéaire de la forme $A^TAx = A^Tb$. Afin de résoudre ce système linéaire, nous utilisons la *formule de Sherman-Morrisson-Woodbury*. La théorie concernant les problèmes aux moindres carrés a été présentée au Chapitre 4. Cependant, nous devons encore introduire la théorie concernant la formule de Sherman-Morrisson-Woodbury.

6.3 Formule de Sherman-Morrison-Woodbury

Afin de résoudre des systèmes linéaires de la forme $\overline{A}x = b$, où $\overline{A} \in \mathbb{R}^{n\times n}$ est non singulière, nous évitons le calcul de l'inverse de la matrice. Ce calcul peut s'avérer très coûteux et rend la matrice instable. Comme nous pouvons le lire dans l'article de l'Université d'Evry [43], nous favorisons donc la factorisation de la matrice $\overline{A}$ en un produit de deux matrices triangulaires :

$$\overline{A} \;=\; LU,$$

où L est triangulaire inférieure et U est triangulaire supérieure. La résolution du système linéaire $\overline{A}x = b$ est alors ramenée à la résolution de deux systèmes triangulaires $Ly = b$ et $Ux = y$. Cependant, si la matrice $\overline{A}$ est non singulière et est le résultat d'une mise à jour de rang un d'une matrice carrée non singulière A tel que :

$$\overline{A} \;=\; A + uv^T,$$

où u, $v \in \mathbb{R}^n$ et $A \in \mathbb{R}^{n\times n}$, nous pouvons exprimer analytiquement l'inverse de cette matrice grâce à la formule de Sherman-Morrison-Woodbury [28] :

$$\overline{A}^{-1} \;=\; A^{-1} - \frac{A^{-1}uv^TA^{-1}}{1 + v^TA^{-1}u}. \tag{6.8}$$

Pour vérifier cette formule, il suffit de multiplier $\overline{A}$ et $\overline{A}^{-1}$ afin de s'assurer que nous obtenons bien l'identité :

$$\begin{aligned}
\overline{A}\,\overline{A}^{-1} &= (A + uv^T)\left(A^{-1} - \frac{A^{-1}uv^TA^{-1}}{1 + v^TA^{-1}u}\right) \\
&= AA^{-1} + uv^TA^{-1} - A\frac{A^{-1}uv^TA^{-1}}{1 + v^TA^{-1}u} - uv^T\frac{A^{-1}uv^TA^{-1}}{1 + v^TA^{-1}u}
\end{aligned}$$

$$
\begin{aligned}
&= AA^{-1} + uv^TA^{-1} - \frac{uv^TA^{-1} + uv^TA^{-1}uv^TA^{-1}}{1 + v^TA^{-1}u} \\
&= AA^{-1} + uv^TA^{-1} - \frac{u\left(1 + v^TA^{-1}u\right)v^TA^{-1}}{1 + v^TA^{-1}u} \qquad (6.9) \\
&= AA^{-1} + uv^TA^{-1} - uv^TA^{-1} \qquad (6.10) \\
&= I_n,
\end{aligned}
$$

où I_n est la matrice identité d'ordre n. De plus, l'égalité (6.10) est obtenue en simplifiant le numérateur et le dénominateur de (6.9), ce qui est possible puisque $v^TA^{-1}u$ est un scalaire.

La formule exprimant l'inverse d'une matrice peut être généralisée à des mises à jour de rang plus élevé. Pour ce faire, considérons U et V, des matrices appartenant à $\mathbb{R}^{n\times p}$, et définissons :

$$\hat{A} = A + UV^T.$$

La formule de Sherman-Morrison-Woodbury [28] devient alors :

$$\hat{A}^{-1} = A^{-1} - A^{-1}U\left(I + V^TA^{-1}U\right)^{-1}V^TA^{-1}. \qquad (6.11)$$

Comme nous l'avons dit ci-dessus, les expressions (6.8) et (6.11) permettent de résoudre des systèmes linéaires de la forme $\overline{A}x = b$ ou $\hat{A}x = b$. La solution x de ces systèmes est obtenue par l'expression :

$$x = A^{-1}b - \frac{A^{-1}uv^TA^{-1}}{1 + v^TA^{-1}u}b,$$

dans le premier cas, et par l'expression :

$$x = A^{-1}b - A^{-1}U\left(I + V^TA^{-1}U\right)^{-1}V^TA^{-1}b,$$

dans le deuxième cas.

6.4 Expression de la mise à jour

Maintenant que nous avons toute la théorie nécessaire, nous pouvons rentrer dans le vif du sujet et obtenir une solution au problème de minimisation de l'algorithme des démons :

$$\min_{u\in\mathbb{R}^d} \frac{1}{2}\left\|F - M\circ(s+u)\right\|^2 + \frac{\sigma_i^2}{2}\left\|u\right\|^2.$$

Afin de résoudre ce problème de minimisation, nous devons le transformer en un problème aux moindres carrés. Pour ce faire, la fonction $f_p : s \mapsto F - M\circ s$ doit être *linéarisée*. Comme nous pouvons le lire dans les articles de T. Vercauteren [45] [47], V. Bieta et T. Möller [4], il existe trois linéarisations possibles. Une première linéarisation est donnée par le Théorème de Taylor avec $t = 0$. En appliquant la formule (4.10), nous obtenons l'expression suivante :

$$
\begin{aligned}
f_p(s+u) &\approx f_p(s) + \nabla f_p(s)^Tu(p) \\
&\approx F(p) - (M\circ s)(p) + \nabla f_p(s)^Tu(p) \\
&\approx F(p) - (M\circ s)(p) - \nabla_p(M\circ s)^Tu(p), \qquad (6.12)
\end{aligned}
$$

où $\nabla f_p(s) = \nabla_p (M \circ s)$ est le gradient de $M \circ s$ par rapport à p. Une deuxième linéarisation est également donnée par le Théorème de Taylor avec $t = 0$. En appliquant la formule (4.11), nous obtenons l'expression suivante :

$$\begin{aligned} f_p(s+u) &\approx f_p(s) + \nabla f_p(s)^T u(p) + \frac{1}{2} u^T(p) \nabla^2 f_p(s) u(p) \\ &\approx f_p(s) + \nabla f_p(s)^T u(p) + \frac{1}{2} \left(\nabla f_p(s+u)^T - \nabla f_p(s)^T \right) u(p) \qquad (6.13) \\ &\approx f_p(s) + \frac{1}{2} \left(\nabla f_p(s)^T + \nabla f_p(s+u)^T \right) u(p) \\ &\approx F(p) - (M \circ s)(p) - \frac{1}{2} \left(\nabla_p (M \circ s)^T + \nabla_p F^T \right) u(p), \qquad (6.14) \end{aligned}$$

où $\nabla f_p(s+u)$ est égal à $\nabla_p F$ puisque le gradient de l'image recalée doit être similaire au gradient de l'image fixe. De plus, $\nabla_p F$ est le gradient de F par rapport à p et l'expression (6.13) est obtenue en appliquant la formule (4.10) du Théorème de Taylor au gradient de f_p avec $t = 0$. La dernière approximation de la fonction f_p est donnée par le Théorème de Taylor avec $t = 1$. Il suffit donc d'appliquer, de nouveau, la formule (4.10) pour obtenir l'expression suivante :

$$\begin{aligned} f_p(s+u) &\approx f_p(s) + \nabla f_p(s+u)^T u(p) \\ &\approx F(p) - (M \circ s)(p) + \nabla f_p(s+u)^T u(p) \\ &\approx F(p) - (M \circ s)(p) - \nabla_p F^T u(p). \qquad (6.15) \end{aligned}$$

Pour chaque linéarisation (6.12), (6.14) et (6.15) définie ci-dessus, nous pouvons remarquer que ce sont les mêmes expressions à un terme près. Afin de faire la suite du développement une seule fois, nous notons A à la place du terme qui diffère d'une linéarisation à l'autre, ce qui donne l'expression écrite ci-dessous :

$$f_p(s+u) \approx F(p) - (M \circ s)(p) + Au(p), \qquad (6.16)$$

où A est de dimension $1 \times d$.

Grâce à cette linéarisation, nous allons montrer que le problème de minimisation de l'algorithme des démons peut être transformé en un problème aux moindres carrés linéaire pour chaque pixel $p \in \Omega$. En effet, l'expression de l'énergie de correspondance (6.7) utilisée dans cet algorithme peut se réécrire de la manière suivante :

$$\begin{aligned} E_s^{corr}(u) &= \frac{1}{2} \left\| F - M \circ (s+u) \right\|^2 + \frac{\sigma_i^2}{2} \left\| u \right\|^2 \\ &= \frac{1}{2} \sum_{p \in \Omega} \left[\left\| F(p) - (M \circ (s+u))(p) \right\|^2 + \sigma_i^2(p) \left\| u(p) \right\|^2 \right] \\ &\approx \frac{1}{2} \sum_{p \in \Omega} \left[\left\| F(p) - (M \circ s)(p) + Au(p) \right\|^2 + \sigma_i^2(p) \left\| u(p) \right\|^2 \right] \\ &\approx \frac{1}{2} \sum_{p \in \Omega} \left\| \begin{pmatrix} F(p) - (M \circ s)(p) + Au(p) \\ \sigma_i(p) I_d u(p) \end{pmatrix} \right\|^2 \\ &\approx \frac{1}{2} \sum_{p \in \Omega} \left\| \begin{pmatrix} A \\ \sigma_i(p) I_d \end{pmatrix} u(p) + \begin{pmatrix} F(p) - (M \circ s)(p) \\ 0 \end{pmatrix} \right\|^2, \qquad (6.17) \end{aligned}$$

où $A \in \mathbb{R}^{1\times d}$, I_d est la matrice identité d'ordre d et 0 est un vecteur de dimension $d \times 1$ contenant uniquement des 0. Nous obtenons ainsi un problème aux moindres carrés linéaire de la forme

$$\min_{x\in\mathbb{R}^n} \quad \frac{1}{2} \| \, Cx + b\|^2 ,$$

avec

$$\begin{aligned} C &= \begin{pmatrix} A \\ \sigma_i(p) I_d \end{pmatrix} \in \mathbb{R}^{(d+1)\times d}, \\ b &= \begin{pmatrix} F(p) - (M \circ s)(p) \\ 0 \end{pmatrix} \in \mathbb{R}^{(d+1)\times 1}, \\ x &= u(p) \in \mathbb{R}^{d\times 1}. \end{aligned}$$

Dans le but de trouver le minimum de la fonction d'énergie, nous devons résoudre, à chaque pixel p, le système des équations normales obtenu en appliquant la formule (4.32) aux éléments C, b et x définis plus tôt, c'est-à-dire :

$$\begin{pmatrix} A^T & \sigma_i(p) I_d \end{pmatrix} \begin{pmatrix} A \\ \sigma_i(p) I_d \end{pmatrix} u(p) \;=\; - \begin{pmatrix} A^T & \sigma_i(p) I_d \end{pmatrix} \begin{pmatrix} F(p) - (M \circ s)(p) \\ 0 \end{pmatrix}.$$

Ce système d'équations normales peut, finalement, s'écrire :

$$\left(A^T A + \sigma_i^2(p) I_d\right) u(p) \;=\; - \left(F(p) - (M \circ s)(p)\right) A^T.$$

Pour résoudre ce système linéaire de la forme $\overline{A}x = b$, nous pouvons calculer l'inverse de la matrice $\overline{A}$ car elle est de la forme $A + uv^T$ avec :

$$A = \sigma_i^2(p) I \overset{not.}{=} \alpha I \ , \quad u = A^T \quad \text{et} \quad v^T = A,$$

où nous avons posé le scalaire $\sigma_i^2(p)$ à α pour plus de facilité dans les calculs qui suivent. Pour calculer l'inverse de cette matrice de la forme $A + uv^T$, nous appliquons donc la formule de Shermann-Morrison-Woodbury (6.8) de la Section 6.3 :

$$\begin{aligned} \left(\alpha I_d + A^T A\right)^{-1} &= \frac{1}{\alpha} I_d - \frac{\frac{1}{\alpha} I_d A^T A \frac{1}{\alpha} I_d}{1 + A \frac{1}{\alpha} I_d A^T} \\ &= \frac{1}{\alpha} I_d - \frac{1}{\alpha^2} \frac{A^T A}{1 + \frac{1}{\alpha} \|A^T\|^2} \\ &= \frac{1}{\alpha} I_d - \frac{1}{\alpha^2} \frac{A^T A}{\frac{\alpha + \|A^T\|^2}{\alpha}} \\ &= \frac{1}{\alpha} I_d - \frac{1}{\alpha} \frac{A^T A}{\alpha + \|A^T\|^2}. \end{aligned} \tag{6.18}$$

Afin d'obtenir l'expression finale de la solution du système des équations normales, nous appliquons l'inverse de la matrice $\left(\alpha I_d + A^T A\right)$ donnée par l'expression (6.18) au second membre $-\left(F(p) - (M \circ s)(p)\right) A^T \overset{not.}{=} -\beta A^T$, c'est-à-dire :

$$\begin{aligned}
u(p) &= \left(\alpha I_d + A^T A\right)^{-1} \left(-\beta A^T\right) \\
&= -\frac{1}{\alpha}\beta A^T + \frac{1}{\alpha}\frac{A^T A}{\alpha + \|A^T\|^2}\beta A^T \\
&= -\frac{\beta}{\alpha}A^T + \frac{\beta}{\alpha}\frac{A^T A A^T}{\alpha + \|A^T\|^2} \\
&= -\frac{\beta}{\alpha}A^T + \frac{\beta}{\alpha}\frac{A^T \left\|A^T\right\|^2}{\alpha + \|A^T\|^2} \\
&= -\frac{\beta}{\alpha}\left[1 - \frac{\left\|A^T\right\|^2}{\alpha + \|A^T\|^2}\right] A^T \\
&= -\frac{\beta}{\alpha}\left[\frac{\alpha + \left\|A^T\right\|^2 - \left\|A^T\right\|^2}{\alpha + \|A^T\|^2}\right] A^T \\
&= -\frac{\beta}{\|A^T\|^2 + \alpha}A^T, \qquad (6.19)
\end{aligned}$$

où nous avons remplacé le scalaire $F(p) - (M \circ s)(p)$ par la notation β. Finalement, si nous remplaçons α et β par leur expression respective dans (6.19), nous obtenons l'expression de mise à jour suivante :

$$u(p) = -\frac{F(p) - (M \circ s)(p)}{\|A^T\|^2 + \sigma_i^2(p)}A^T. \qquad (6.20)$$

Lors de l'étape de linéarisation, nous avons obtenu trois approximations différentes de la fonction $f_p : \; s \mapsto F - M \circ s$. Nous avions alors remplacé par A le terme qui diffère dans chaque expression. En appliquant le changement inverse dans (6.20), nous obtenons donc trois expressions pour le vecteur de déplacement additionnel. De plus, en supposant que

$$\sigma_i^2(p) = \mid F(p) - (M \circ s)(p) \mid,$$

nous avons les trois *vecteurs de déplacement* décrits ci-dessous :

$$u(p) = -\frac{F(p) - (M \circ s)(p)}{\|\nabla_p (M \circ s)\|^2 + \left(F(p) - (M \circ s)(p)\right)^2}\nabla_p (M \circ s), \qquad (6.21)$$

$$\begin{aligned}
u(p) &= -\frac{F(p) - (M \circ s)(p)}{\left\|\frac{1}{2}\left(\nabla_p F + \nabla_p (M \circ s)\right)\right\|^2 + \left(F(p) - (M \circ s)(p)\right)^2}\left(\frac{1}{2}\left(\nabla_p F + \nabla_p (M \circ s)\right)\right) \\
&= -\frac{2\left(F(p) - (M \circ s)(p)\right)}{\|\nabla_p F + \nabla_p (M \circ s)\|^2 + \left(F(p) - (M \circ s)(p)\right)^2}\left(\nabla_p F + \nabla_p (M \circ s)\right), \qquad (6.22)
\end{aligned}$$

$$u(p) = -\frac{F(p) - (M \circ s)(p)}{\|\nabla_p F\|^2 + \left(F(p) - (M \circ s)(p)\right)^2}\nabla_p F. \qquad (6.23)$$

L'équation (6.22) est l'expression de la mise à jour utilisée dans la version de J.-P. Thirion de l'algorithme des démons [4]. L'équation (6.23), quand-à elle, est l'expression de la mise à jour utilisée dans la version de T. Vercauteren de l'algorithme des démons [45]. Ce sont ces deux versions qui sont implémentées dans la librairie ITK et qui vont nous intéresser dans la suite de ce chapitre.

Cependant, il existe une incompatibilité dans les unités entre les deux termes du dénominateur des différentes expressions. L'unité du premier terme est intensité^2/mm^2 alors que l'unité du second terme est intensité^2. C'est pourquoi nous normalisons le second terme par un facteur K dans les implémentations de la librairie ITK. Ce facteur est la *valeur quadratique moyenne* de la taille des pixels le long de chaque axe. La notion de taille des pixels a été présentée dans la Section 2.2 du Chapitre 2 sous le nom de "spacing". En normalisant le second terme du dénominateur, nous obtenons les expressions suivantes pour les vecteurs de déplacement :

$$u(p) = -\frac{F(p) - (M \circ s)(p)}{\|\nabla_p (M \circ s)\|^2 + \frac{(F(p) - (M \circ s)(p))^2}{K}} \nabla_p (M \circ s), \tag{6.24}$$

$$u(p) = -\frac{2\left(F(p) - (M \circ s)(p)\right)}{\|\nabla_p F + \nabla_p (M \circ s)\|^2 + \frac{(F(p) - (M \circ s)(p))^2}{K}} \left(\nabla_p F + \nabla_p (M \circ s)\right), \tag{6.25}$$

$$u(p) = -\frac{F(p) - (M \circ s)(p)}{\|\nabla_p F\|^2 + \frac{(F(p) - (M \circ s)(p))^2}{K}} \nabla_p F. \tag{6.26}$$

6.5 Librairie ITK

Plusieurs méthodes de recalage sont implémentées dans la librairie ITK. Comme ces méthodes utilisent parfois les mêmes fonctions, elles ne sont pas implémentées à chaque fois. Les fonctions se trouvent une seule fois dans la librairie mais peuvent être appelées à plusieurs endroits différents. Le but de cette section est de comprendre l'implémentation de l'algorithme des démons. Pour ce faire, nous devons d'abord comprendre comment fonctionne la librairie ITK et, plus précisément, où trouver les différentes fonctions considérées dans notre cas.

Les algorithmes de recalage avec déformations s'appuient sur le framework "Finite Difference Solver" (`\Modules\Core\FiniteDifference\include`) de ITK. Ce framework est un ensemble de classes permettant de résoudre des équations différentielles partielles sur les images avec un processus de mise à jour itératif et fini. L'algorithme général est implémenté dans la méthode "GenerateData()" de "FiniteDifferenceImageFilter" et est de la forme suivante [12] :

WHILE NOT convergence :

FOR ALL pixels p :

time_step = calculate_change(p),

udate(p, time_step).

L'équation générale sous-jacente qui décrit la mise à jour $n+1$ au pixel p sur l'image u est la suivante :

$$u_p^{n+1} = u_p^n + \triangle u_p^n \triangle t.$$

Si nous partons de la méthode "GenerateData()" implémentée dans "FilterDifferenceImageFilter" de la librairie ITK, nous trouvons l'équivalent du pseudo-code écrit ci-dessous :

Algorithme 6.2

```
FiniteDifferenceImageFilter : : GenerateData()
{
    [...]
    Initialize() ;
    // Crée la structure qui contiendra les résultats des calculs de mise à jour
    AllocateUpdateBuffer() ;
    [...]
    // Arrêt de l'algorithme lorsque le résultat rencontre le critère d'arrêt
    while ( ! Halt() )
    {
        // Initialisation spécifique à l'itération
        InitializeIteration() ;
        // Calcul de la mise à jour
        dt = CalculateChange() ;
        // Application de la mise à jour
        ApplyUpdate(dt) ;
    }
    [...]
}
```

Dans l'Algorithme 6.2, nous remarquons qu'il n'y a aucune implémentation, c'est une méthode abstraite. Dans ce cas, les fonctions sont juste déclarées dans la classe et l'implémentation concrète se trouve soit dans cette même classe, soit dans une sous-classe. Pour comprendre dans quelles classes se trouve l'implémentation spécifique de ces fonctions, il faut suivre l'enchaînement des appels entre les classes comme illustré sur la Figure 6.4. Chaque classe encadrée en rouge sur cette figure gère une partie du problème. Par exemple, la fonction "AllocateUpdateBuffer()" est implémentée dans la sous-classe "DenseFiniteDifferenceImageFilter", la fonction "Halt()" est implémentée dans la classe principale "FiniteDifferenceImageFilter", la fonction "InitializeIteration()" est implémentée dans la sous-classe "PDEDeformableRegistrationFilter", les fonctions "CalculateChange()" et "ApplyUpdate()" sont implementées dans la sous-classe "DenseFiniteDifferenceImageFilter".

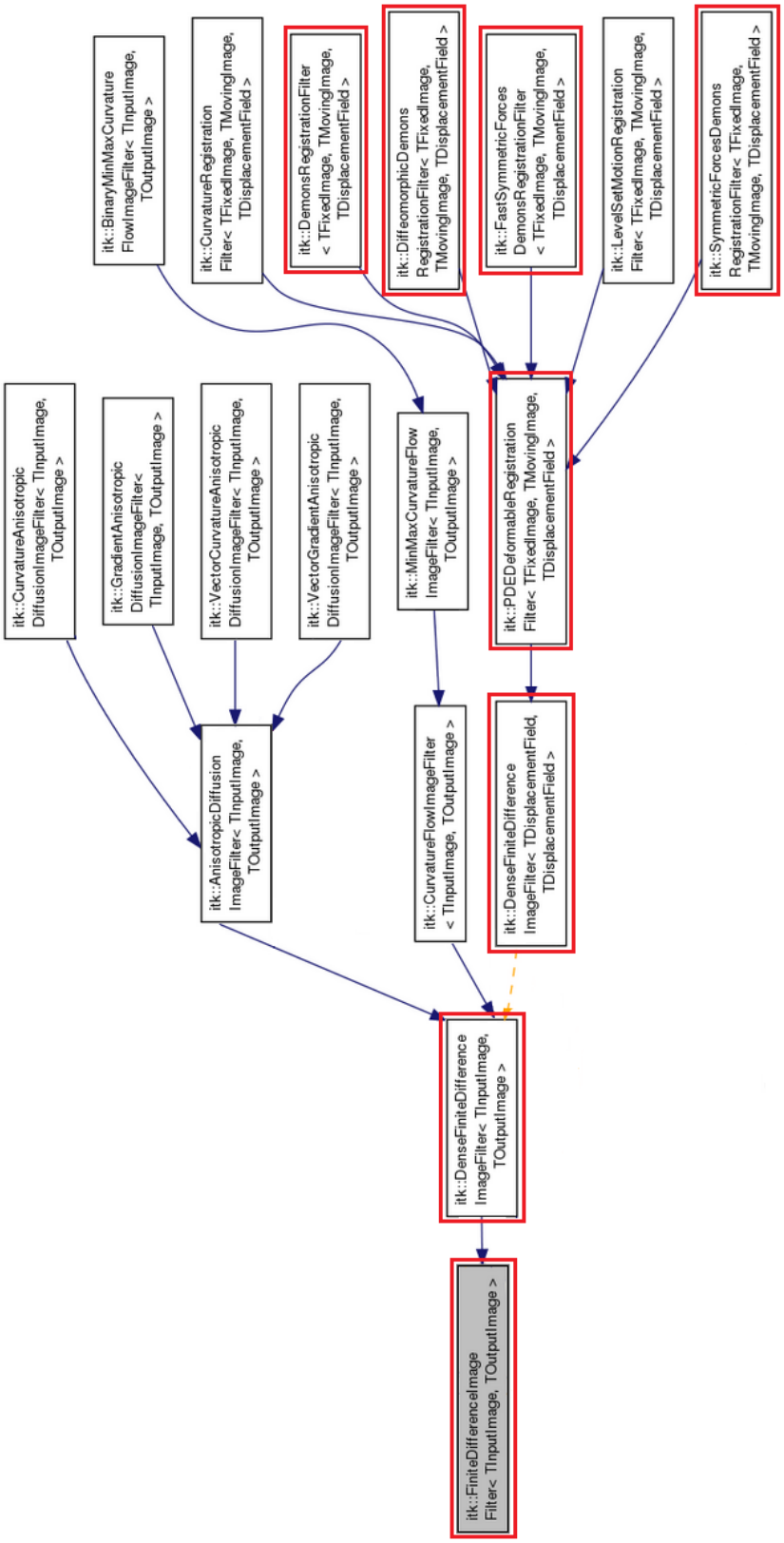


FIGURE 6.4 – Enchaînement des classes utilisées dans les algorithmes de recalage [12]

Les méthodes illustrées à la Figure 6.4 ne peuvent être considérées seules car elles constituent le composant "solver" qui ne gère que la boucle générale de l'algorithme et les images d'input et d'output. A chacune de ces méthodes correspond donc un composant "function" qui gère les calculs localement au voisinage des pixels dans les images. Dans l'approche du recalage par les démons, l'héritage depuis l'algorithme général jusqu'à l'implémentation concrète passe par différentes classes comme le résume la Table 6.1. Comme nous l'avons déjà dit auparavant, l'implémentation générale est faite dans la classe du premier niveau, ensuite, plus l'implémentation est spécifique, plus elle est faite à bas niveau.

	Solver
1	itkFiniteDifferenceImageFilter
2	itkDenseFiniteDifferenceImageFilter
3	itkPDEDeformableRegistrationFilter
4	itkDemonsRegistrationFilter itkDiffeomorphicDemonsRegistrationFilter itkSymmetricForcesDemonsRegistrationFilter FastSymmetricForcesDemonsRegistrationFilter

	Function
1	itkFiniteDifferenceImageFunction
2	itkDenseFiniteDifferenceImageFunction
3	itkPDEDeformableRegistrationFunction
4	itkDemonsRegistrationFunction itkDiffeomorphicDemonsRegistrationFunction itkSymmetricForcesDemonsRegistrationFunction FastSymmetricForcesDemonsRegistrationFunction

TABLE 6.1 – Héritage des classes utilisées dans les algorithmes de recalage

Les fichiers cités dans la Table 6.1 se trouvent dans différents dossiers de la librairie ITK. Dans certains cas, il n'existe pas d'objet "Function" ou d'objet "Filter" et dans d'autres cas, l'objet "Function" correspond à l'objet "Filter". Voici la localisation des fichiers dans ITK (version 4.4.2) :

- itkFiniteDifferenceImage

  Filter = Function : `\Modules\Core\FiniteDifference\include`
- itkDenseFiniteDifferenceImage
  - Filter : `\Modules\Core\FiniteDifference\include`
  - Function : /
- itkPDEDeformableImageRegistration
  - Filter : `\Modules\Registration\PDEDeformable\include`
  - Function : `\Modules\Registration\Common\include`

- itkDemonsRegistration

  Filter = Function : `\Modules\Registration\PDEDeformable\include`
- itkSymmetricForcesDemonsRegistration

  Filter = Function : `\Modules\Registration\PDEDeformable\include`
- itkFastSymmetricForcesDemonsRegistration
  - Filter : /
  - Function : `\Modules\Registration\PDEDeformable\include`
- itkDiffeomorphicDemonsRegistration
  - Filter : `\Modules\Nonunit\Review\include`
  - Function : /

L'empilage des classes au niveau 4 de la Table 6.1 ne correspond pas à un héritage au sens informatique ni au sens mathématique. Du point de vue informatique, elles sont sur le même niveau, ce sont les implémentations les plus spécifiques. Au niveau de la théorie mathématique, "itkSymmetricForcesDemonsRegistrationFilter" correspond à l'algorithme proposé par J.-P. Thirion, "itkFastSymmetricForcesDemonsRegistrationFilter" en est une optimisation où on garde une copie déformée de l'image mouvante pour l'évaluation du gradient, "itkDemonsRegistrationFilter" et "itkDiffeomorphicDemonsRegistrationFilter" correspondent à l'algorithme proposé par T. Vercauteren. De plus, "itkDiffeomorphicDemonsRegistrationFilter" se base sur "itkDemonsRegistrationFilter" à la différence qu'il utilise des difféomorphismes exponentiels comme fonction de mise à jour, c'est pourquoi il n'existe pas d'objet "Function".

Les étapes clés de l'algorithme des démons (Algorithme 6.1) sont pilotées depuis la boucle générale de la méthode "GenerateData()" (Algorithme 6.2). Les implémentations de ces étapes se trouvent dans les différentes classes citées dans la Table 6.1. Notons que certaines classes sont overridées, c'est-à-dire ré-implémentées dans les sous-classes pour faire des traitements plus spécifiques. C'est, par exemple, le cas de "ApplyUpdate()". Dans la suite se trouvent les grandes étapes de l'Algorithme 6.2 et le cheminement à suivre pour trouver les méthodes où ces étapes sont implémentées. Lorsqu'une méthode fait appel à une autre, nous l'indiquerons par une flèche (méthode 1 $\Rightarrow$ méthode 2).

- **Point d'entrée de l'algorithme :** FiniteDifferenceImageFilter : : GenerateData().
- **Initialisation du vecteur déplacement à 0 :** PDEDeformableRegistrationFilter : : Initialize().
- **Calcul du gradient :** DenseFiniteDifferenceImageFilter : : AllocateUpdateBuffer().
- **Condition d'arrêt :** FiniteDifferenceImageFilter : : Halt().
- **Calcul de l'intensité dans l'image déformée :** PDEDeformableRegistrationFilter : : InitializeIteration().
- **Calcul des changements :** DenseFiniteDifferenceImageFilter : : CaculateChange() $\Rightarrow$
  DenseFiniteDifferenceImageFilter : : CalculateChangeThreaderCallback() $\Rightarrow$
  DenseFiniteDifferenceImageFilter : : ThreadedCalculateChange() $\Rightarrow$

- DemonsRegistrationFunction : : ComputeUpdate().
- SymmetricForcesDemonsRegistrationFunction : : ComputeUpdate().
- FastSymmetricForcesDemonsRegistrationFunction : : ComputeUpdate().
- Les implémentations spécifiques se trouvent dans les classes de niveau 4 dans la hiérarchie présentée dans la Table 6.1.

- **Application de la mise à jour :**
  - **Approche générale (non spécifique à la méthode des démons) :**
    DenseFiniteDifferenceImageFilter : : ApplyUpdate() ⇒
    DenseFiniteDifferenceImageFilter : : ApplyUpdate() ⇒
    - ⋆ DenseFiniteDifferenceImageFilter : : ThreadedApplyUpdate().
    - ⋆ Cette méthode contient la boucle qui applique localement les mises à jours.
  - **Méthode des démons :**
    Ré-implémentation dans les classes de niveau 4, par exemple :
    - ⋆ SymmetricForcesDemonsRegistrationFilter : : ApplyUpdate().
    - ⋆ Dans cette méthode, nous retrouvons les étapes de lissage et l'application de la mise à jour standard (appel à la classe supérieure DenseFiniteDifferenceImage-Filter où se trouve l'implémentation).

Comme nous l'avons énoncé ci-dessus, les implémentations spécifiques de la mise à jour se trouvent dans les classes de niveaux 4. Dans un premier temps, nous nous intéressons à la version de l'algorithme des démons proposée par J.-P. Thirion qui est implémentée dans "itkSymmetric-ForcesDemonsRegistrationFilter". Vérifions si l'Algorithme 6.3 correspond à l'expression (6.25) trouvée auparavant et rappelée ci-après :

$$u(p) \;=\; -\frac{2\left(F(p)-(M\circ s)(p)\right)}{\left\|\nabla_p F+\nabla_p(M\circ s)\right\|^2+\dfrac{\left(F(p)-(M\circ s)(p)\right)^2}{K}}\left(\nabla_p F+\nabla_p(M\circ s)\right).$$

Algorithme 6.3

SymmetricForcesDemonsRegistrationFunction : : ComputeUpdate()
{

```
[...]
// Calcul de ‖∇_p F + ∇_p(M ∘ s)‖²
fixedPlusMovingGradientSquaredMagnitude = 0
for ( dim = 0 ; dim < ImageDimension ; dim++ )
{
    fixedPlusMovingGradientSquaredMagnitude += sqr(fixedGradient[dim] + movingGradient[dim]) ;
}
// Calcul de (F(p) − (M ∘ s)(p))
```

speedValue = fixedValue − movingValue ;

// Calcul du dénominateur de la forme $\|\nabla_p F + \nabla_p (M \circ s)\|^2 + \frac{(F(p) - (M \circ s)(p))^2}{K}$

denominator = sqr(speedValue) / m_Normalizer + fixedPlusMovingGradientSquaredMagnitude ;

[...]

for (j = 0 ; j < ImageDimension ; j++)

{

// Cacul de l'expression finale de la mise à jour

update[j] = 2 ∗ speedValue ∗ (fixedGradient[j] + movingGradient[j]) / denominator ;

}

[...]

}

Dans l'Algorithme 6.3, nous retrouvons bien toutes les étapes permettant de calculer l'expression de la mise à jour. La seule différence entre la pratique et la théorie est le signe lors du calcul de l'expression finale. Cependant, cette différence est équilibrée lors de l'application de la mise à jour dans la méthode "ApplyUpdate()". Le facteur de normalisation, nommé "m_Normalizer", est implémenté dans la méthode "InitializeIteration()" de cette même classe. Rappelons que ce facteur de normalisation est la valeur quadratique moyenne de l'espacement des pixels et est de la forme :

$$K = \frac{1}{N} \sum_{i=0}^{N} S_i^2, \tag{6.27}$$

où N est la dimension de l'image et S est le *spacing*, c'est-à-dire un vecteur contenant la taille des pixels le long de chaque axe. L'Algorithme 6.4 permet de calculer ce facteur :

Algorithme 6.4

SymmetricForcesDemonsRegistrationFunction : : InitializeIteration()

{

[...]

// Calcul de $\sum_{i=0}^{N} S_i^2$

m_Normalizer = 0.0 ;

for (k = 0 ; k < ImageDimension ; k++)

{

m_Normalizer + = m_FixedImageSpacing[k] ∗ m_FixedImageSpacing[k] ;

}

// Division par N du facteur "m_Normalizer"

```
    m_Normalizer /= ImageDimension;
    [...]
}
```

Dans le code décrit ci-dessus, la variable "ImageDimension" désigne la dimension de l'image, notée N dans la formule (6.27).

Intéressons-nous maintenant à la version de l'algorithme des démons proposée par T. Vercauteren qui est implémentée dans "itkDemonsRegistrationFilter". Vérifions si l'Algorithme 6.5 correspond à l'expression (6.26) trouvée auparavant et rappelée ci-après :

$$u(p) = -\frac{F(p) - (M \circ s)(p)}{\|\nabla_p F\|^2 + \frac{(F(p) - (M \circ s)(p))^2}{K}} \nabla_p F.$$

Algorithme 6.5

DemonsRegistrationFunction : : ComputeUpdate()

```
{
    [...]
    // Calcul de ‖∇_p F‖²
    gradientSquaredMagnitude = 0;
    for ( j = 0; j < ImageDimension; j++ )
    {
        gradientSquaredMagnitude += sqr(gradient[j]);
    }
    // Calcul de (F(p) − (M ∘ s)(p))
    speedValue = fixedValue − movingValue;
    sqr_speedValue = sqr(speedValue);
    [...]
    // Calcul du dénominateur de la forme ‖∇_p F‖² + (F(p) − (M ∘ s)(p))² / K
    denominator = sqr_speedValue / m_Normalizer + gradientSquaredMagnitude;
    [...]
    for ( j = 0; j < ImageDimension; j++ )
    {
        // Cacul de l'expression finale de la mise à jour
        update[j] = speedValue * gradient[j] / denominator;
        [...]
    }
    [...]
```

```
}
```

De nouveau, nous retrouvons les étapes qui permettent de calculer l'expression de la mise à jour. De plus, le calcul du facteur de normalisation "m_Normalize" est implémenté dans la méthode "InitializeIteration()" qui se trouve dans cette même classe. Nous ne rappelons pas le code qui permet de calculer ce facteur car il est identique à celui décrit auparavant (Algorithme 6.4).

Que ce soit la version de l'algorithme des démons proposée par J.-P. Thirion ou celle proposée par T. Vercauteren, nous retrouvons toutes les étapes de l'Algorithme 6.1 dans la librairie ITK. Du point d'entrée de l'algorithme jusqu'à l'application de la mise à jour en passant par le calcul de cette mise à jour, nous avons trouvé les méthodes où sont implémentées ces étapes.

Chapitre 7

Conclusion et perspectives

Au cours de ce mémoire, nous avons abordé la problématique de l'imagerie médicale et, plus précisément, du recalage d'images médicales. Comme nous l'avons expliqué, il s'agit d'un processus de traitement d'images qui consiste en la mise en correspondance d'images afin de pouvoir comparer ou combiner leurs informations. Le processus de recalage est constitué de plusieurs étapes dont l'optimisation qui permet de minimiser la distance entre les deux images à recaler. Cette distance est caractérisée par un critère de similarité. Dans ce domaine, de nombreuses méthodes d'optimisation sont utilisées comme la méthode de plus forte pente ou la méthode de Gauss-Newton.

La librairie ITK, outil informatique exploité par le CHU de Mont-Godinne, fournit un ensemble de codes C++ permettant de recaler deux images. Parmi ces codes, il existe deux implémentations de la méthode de plus forte pente qui est la méthode d'optimisation la plus utilisée dans le recalage d'images. Après avoir introduit la théorie concernant cette méthode, nous avons étudié la correspondance entre la théorie et l'implémentation. Suite à cette étude, nous avons pu retranscrire les deux méthodes en pseudo-code, ce qui nous donne les Algorithmes 5.1 et 5.2. Quel que soit l'algorithme considéré, aucune recherche linéaire n'est appliquée pour trouver la longueur du pas. Cependant, de manière générale, le code se rapproche fortement de la méthode de base.

Pour recaler deux images, le CHU utilise, plus particulièrement, la méthode des démons proposée par J.-P. Thirion [42]. Dans ce cas, aucune méthode d'optimisation implémentée dans la librairie ITK n'est utilisée. En effet, l'optimisation est directement gérée dans la boucle générale de l'algorithme des démons. Comme le montre l'Algorithme 6.1 de la méthode des démons, il faut minimiser l'énergie de correspondance afin de trouver l'expression de la mise à jour du vecteur de déplacement. Comme ce problème de minimisation est un problème aux moindres carrés, nous avons utilisé la méthode de Gauss-Newton pour le résoudre. De cette manière, nous avons trouvé l'expression finale de la mise à jour. Ensuite, nous avons fait le lien entre les grandes étapes de l'algorithme des démons (Algorithme 6.1) et leur implémentation dans la librairie ITK. Grâce à cette mise en correspondance, nous savons où se trouve chaque étape de l'implémentation de cette méthode dans la librairie. Pour finir, nous avons étudié les codes permettant de calculer l'expression de la mise à jour afin de s'assurer que l'implémentation correspondait bien à l'expression obtenue précédemment.

Dans le cadre des collaborations existantes entre l'Université de Namur et le CHU de Mont-Godinne, trois étudiants de la Faculté d'Informatique ont déjà travaillé dans le domaine de l'imagerie médicale. Cependant, ce mémoire, qui étudie le domaine de l'imagerie médicale et du recalage d'images médicales, est le premier travail réalisé par un mathématicien. Selon les membres du service de Médecine Nucléaire du CHU, la méthode de recalage des démons demande un temps de compilation très important. La prochaine étape serait donc de faire tourner ces codes afin de voir quelle(s) étape(s) du processus est/sont trop lente(s). Pour ce faire, il est possible de mettre des "flags" dans les codes. De manière générale, les perspectives sont d'améliorer le temps de calcul des codes, d'optimiser l'exécution et, éventuellement, d'adapter les techniques d'optimisation utilisées dans la méthode des démons.

Liste des tableaux

Table des figures

Bibliographie

[1] Atif J., *Recalage non-rigide multimodal des images radiologiques par information mutuelle quadratique normalisée*, Université de Paris XI - Orsay, 2004

[2] Benniss F.,Laraqui Bensouda N. et Fathi K., *Centre Radiologie Abdelmoumen*, `http://www.radiologieabdelmoumen.com`, 2011, consulté le 17 mars 2013

[3] Bierlaire M., *Introduction à l'optimisation différentiable*, Presses Polytechniques et Universitaires Romandes (PPUR), 2006

[4] Bieta V. et Möller T., *A Short Note on Thirions Demons Algorithm*, Southest Asian Bulletin of Mathematics, 2013

[5] Bloch I., *Recalage d'images 2D et 3D*, `http://perso.telecom-paristech.fr/~bloch/VOIR/recalageVOIR.pdf`, -, consulté le 12 avril 2013

[6] Bonnet P., *Cours de traitement d'images*, `http://www-lagis.univ-lille1.fr/~bonnet/image/OpGeo.pdf`, -, consulté le 12 avril 2013

[7] Calande S., *Détection de tumeurs dans la région pulmonaire sur base d'acquisitions PET/CT syncronisées*, Mémoire de fin d'étude, Université de Namur, 2009

[8] Inc. Kitware, *Cross-platform Make (CMake)*, `http://www.cmake.org/`, 2013, consulté le 7 novembre 2013

[9] Cohen M. et al., *Laboratoire d'imagerie fonctionnelle*, `http://www.imed.jussieu.fr/fr/equipes/e2/onco_recal_temp_irm/onco_recal_temp_irm.php`, 2013, consulté le 8 avril 2013

[10] DicomWorks, *Free DICOM software*, `http://www.dicomworks.com/`, -, consulté le 1er mai 2013

[11] Dojat M., *Traitement d'Images pour l'Imagerie Fonctionnelle Cérébrale*, `http://nifm.ujf-grenoble.fr/~dojatm/cours/TraitDataIrmf.pdf`, -, consulté le 12 avril 2013

[12] Doxygen, *Insight Segmentation and Registration Toolkit*, `http://www.itk.org/Doxygen/html/classitk_1_1FiniteDifferenceImageFilter.html`, 2014, consulté le 12 avril 2014

[13] Gilbert J.C., Conditions d'optimalité, `https://who.rocq.inria.fr/Jean-Charles.Gilbert/ensta/04-co.pdf`, 2010, consulté le 12 main 2013

[14] Gouarin L., *Formation en Calcul Scientifique - Introduction à CMake*, `http://calcul.math.cnrs.fr/Documents/Ecoles/LEM2I/Mod2/tp_cmake.pdf`, 2011, consulté le 1er mai 2014

[15] Gitorious AS, *QT Project Hosting (QT)*, `http://qt-project.org/`, 2013, consulté le 16 mars 2014

[16] Gouarin L., *Introduction à CMake*, Formation en Calcul scientifique, 2011

[17] Goujeon F., *Club des développeurs et IT pro; initiation à CMake*, http://florian-goujeon.developpez.com/cours/cmake/initiation/, 2013, consulté le 18 novembre 2013

[18] Groupe MathéTIC, *Courbure d'une courbe*, http://www.lmrl.lu/mathematiques/TI/1re/B%20uniquement/Courbure%202B.pdf, 2004, consulté le 9 avril 2013

[19] Hill D.L.G., Batchelor P.G., Holden M et Hawkes D.J., *Medical image registration*, Institute of Physics Publishing, London, 2000

[20] Inc. Kitware, *Visualization Toolkit (VTK)*, http://www.vtk.org/, 2013, consulté le 5 novembre 2013

[21] Inc. Kitware, *Insight Segmentation and registration Toolkit (ITK)*, http://www.itk.org/, 2013, consulté le 5 novembre 2013

[22] Jan S., *De la physique nucléaire à l'imagerie médicale*, http://clrwww.in2p3.fr/semi/2005/sources/20050211.pdf, 2005, consulté le 7 avril 2013

[23] Luis I. et al., *The ITK Software Guide, Updated for ITK version 2.4*, Second Edition, 2005

[24] Malandain G., *Mesures de similarité pour le recalage multimodal*, http://www-sop.inria.fr/epidaure/personnel/malandain/cours/gt-recalage-multimodal/, 2002, consulté le 13 mars 2013

[25] MathWorks, *MATLAB : The Language of Technical Computing*, http://www.mathworks.nl/products/matlab/index.html, 2014, consulté le 16 mai 2014

[26] MeVis Medical Solutions, *Medical image processing and visualization (MeVisLab)*, http://www.mevislab.de/, 2013, consulté le 7 nombre 2013

[27] Michiels Y., *Amélioration d'un outil de traitement pour la segmentation d'images médicales 2D et 3D dans le domaine de l'ostéoarticulaire*, Mémoire de fin d'étude, Université de Namur, 2012

[28] Nocedal J. et Wright S. J., *Numerical Optimization*, Springer Series in Operations Research, 1999, p.605

[29] Owens J. D. et al, *Fast Deformable Registration on the GPY : A CUDA Implementaion of Demons*, http://www.idav.ucdavis.edu/~pmuyan/publications/ICSSA-2008-Fast_Deformable_Registration.pdf, 2008, consulté le 4 mai 2014

[30] Petitjean C., *Le recalage d'images*, http://carolinepetitjean.free.fr/enseignements/ti/coursrecalage_petitjean.pdf, 2012, consulté le 9 avril 2013

[31] Public Wiki, *ITK/Examples/Included/Registration*, http://www.itk.org/Wiki/ITK/Examples/Included/Registration, 2013, consulté le 16 mars 2014

[32] Printems J. et Rousse V., *Méthode de Newton, systèmes dynamiques et optimisation*, http://perso-math.univ-mlv.fr/users/printems.jacques/L2/2008-09/notes_cours_newton.pdf, 2008, consulté le 12 mai 2013

[33] Richard F., *Recalage d'images biomédicales*, http://www.latp.univ-mrs.fr/~richard/Presentations/lena06.pdf, -, consulté le 12 avril 2013

[34] Risser L., *Recalage d'image par minimisation de l'information mutuelle*, http://laurent.risser.free.fr/FAC/rapportStageMaitrise.pdf, 2002, consulté le 5 mai 2013

[35] Roche A., *Recalage d'images médicales par inférence statistique*, Université de Nice-Sophia Antipolis, France, 2001

[36] Sam et Max, *Ajouter un chemin à la variable d'environnement PATH sous Windows*, http://sametmax.com/ajouter-un-chemin-a-la-variable-denvironnement-path-sous-windows/, 2013, consulté le 18 novembre 2013

[37] Sarrut D., et al., *MetaImage*, http://www.creatis.insa-lyon.fr/rio/MetaImage, 2008, consulté le 11 avril 2013

[38] Sartenaer A., *On trust-region methods in nonlinear optimization*, University of Namur, Belgium, 2006

[39] Seda S., *Introduction à la norme DICOM et l'extension DICOM-RT*, http://www.creatis.insa-lyon.fr/~dsarrut/mybib/2004/rapportSeda2004.pdf, 2004, consulté le 10 avril 2013

[40] Strodiot J.-J., *Introduction to optimization*, University of Namur, Belgium, 2011

[41] Têtes chercheuses, *Lumières médicales*, http://www.tetes-chercheuses.fr/magazines/numero-5/dossier/la-medecine-nucleaire-271/, 2007, consulté le 8 avril 2013

[42] Thirion J.-P., *Image matching as a diffusion process : an analogy with Maxwell's demons*, http://hal.inria.fr/docs/00/61/50/88/PDF/thirion-media-1998.pdf, INRIA Sophia Antipolis, France, 2004, consulté le 11 mai 2014

[43] University d'Evry, *Analyse numérique matricielle - Élimination de Gauss, factorisation LU et applications*, http://www.maths.univ-evry.fr/pages_perso/valexandre/L3MAN-TP1.pdf, 2008, consulté le 1er mai 2014

[44] University *of* Maryland Medical Center, *PET/CT*, www.umm.edu/petct, 2012, consulté le 13 mars 2013

[45] Vercauteren T. et al., *Non-parametric Diffeomorphic Image Registration with the Demons Algorithm*, http://www-sop.inria.fr/asclepios/Publications/Tom.Vercauteren/DiffeoDemons-MICCAI07-Vercauteren.pdf, 2007, consulté le 4 mai 2014

[46] Vercauteren T. et al., *Symmetric Log-Domain Diffeomorphic Registration : A Demons-based Approach*, http://link.springer.com/chapter/10.1007/978-3-540-85988-8_90, 2008, consulté le 4 mai 2014

[47] Vercauteren T. et al., *Diffeomorphic Demons : Efficient Non-parametric Image Registration*, http://hal.archives-ouvertes.fr/docs/00/34/96/00/PDF/DiffeoDemons-NeuroImage08-Vercauteren.pdf, 2008, consulté le 5 mai 2014

[48] Wiki David, *MetaImage (format de fichiers)*, http://dlegland.wikispaces.com/MetaImage+(format+de+fichiers), 2013, consulté le 11 avril 2013

[49] Wikipédia, *Anatomie humaine*, http://fr.wikipedia.org/wiki/Anatomie_humaine, 2013, consulté le 1er mai 2013

[50] Wikipédia, *Cmd*, http://fr.wikipedia.org/wiki/Cmd, 2013, consulté le 18 novembre 2013

[51] Wikipédia, *Déformation élastique*, http://fr.wikipedia.org/wiki/D%C3%A9formation_%C3%A9lastique#Cisaillement, 2014, consulté le 25 avril 2014

[52] Wikipédia, *Digital imaging and communications in medicine*, http://fr.wikipedia.org/wiki/Digital_imaging_and_communications_in_medicine, 2013, consulté le 10 avril 2013

[53] Wikipédia, *Imagerie médicale*, http://http://fr.wikipedia.org/wiki/Imagerie_médicale, 2013, consulté le 7 avril 2013

[54] Wikipédia, *Loi normale multidimensionnelle*, http://fr.wikipedia.org/wiki/Loi_normale_multidimensionnelle, 2014, consulté le 2 mai 2014

[55] Wikipédia, *Métabolisme*, http://fr.wikipedia.org/wiki/Métabolisme, 2013, consulté le 1er mai 2013

[56] Wikipédia, *MeVisLab*, http://en.wikipedia.org/wiki/MeVisLab, 2014, consulté le 12 avril 2014

[57] Wikipédia, *Physiologie*, http://fr.wikipedia.org/wiki/Physiologie, 2013, consulté le 1er mai 2013

[58] Wikipédia, *Recalage d'images*, http://fr.wikipedia.org/wiki/Recalage_d'images, 2013, consulté le 13 mars 2013

[59] Wikipédia, *Système de référence en anatomie*, http://fr.wikipedia.org/wiki/Système_de_référence_en_anatomie, 2013, consulté le 8 avril 2013

[60] Wikipédia, *Traitement d'images*, http://fr.wikipedia.org/wiki/Traitement_d'images, 2014, consulté le 2 mai 2014

[61] Wilfart S., *Développement d'un outil de traitement spécialisé pour la segmentation d'image médicale 3D et validation dans le domaine de l'ostéoartucilaire*, Mémoire de fin d'étude, Université de Namur, 2011

[62] Youbing Y., *Mass preserving nonrigid registration of CT lung using cubic B-spline*, Medical Physics, Iowa, 2009

Annexe A

Codes source des méthodes d'optimisation

Dans cette annexe se trouvent les codes source des méthodes d'optimisation qui ont été retranscrits en pseudo-code dans le Chapitre 5. Pour rappel, ces codes viennent du dossier "Optimizer" de la librairie ITK [21] et illustrent deux variantes de la méthode de descente du gradient. L'Algorithme 5.1 correspond à la méthode de descente du gradient avec un pas régulier (Section A.1), dont le code source se nomme "itkRegularStepGradientBaseOptimizer.cxx". Le calcul de la mise à jour de l'Algorithme 5.1 se trouve dans un autre code source nommé "itkRegularStepGradientDescentOptimizer.cxx" (Section A.2). L'Algorithme 5.2 correspond à la méthode de descente du gradient (Section A.3), dont le code source se nomme "itkGradient-DescentOptimizer.cxx".

A.1 Méthode de descente du gradient avec un pas régulier

```
{
/** Constructor **/

RegularStepGradientDescentBaseOptimizer
::RegularStepGradientDescentBaseOptimizer()
{
  m_MaximumStepLength = 1.0;
  m_MinimumStepLength = 1e-3;
  m_GradientMagnitudeTolerance = 1e-4;
  m_NumberOfIterations = 100;
  m_CurrentIteration   =   0;
  m_Value = 0;
  m_Maximize = false;
  m_CostFunction = 0;
  m_CurrentStepLength   =   0;
  m_StopCondition = Unknown;
  m_Gradient.Fill(0.0f);
  m_PreviousGradient.Fill(0.0f);
  m_RelaxationFactor = 0.5;
  m_StopConditionDescription.str("");
}
```

```
/** Start the optimization **/
void
RegularStepGradientDescentBaseOptimizer
::StartOptimization(void)
{
  m_CurrentStepLength         = m_MaximumStepLength;
  m_CurrentIteration          = 0;

  m_StopCondition = Unknown;
  m_StopConditionDescription.str("");
  m_StopConditionDescription << this->GetNameOfClass() << ": ";

  // validity check for the value of GradientMagnitudeTolerance
  if ( m_GradientMagnitudeTolerance < 0.0 )
    {
    itkExceptionMacro(<< "Gradient magnitude tolerance must be"
                      "greater or equal 0.0. Current value is "
                      << m_GradientMagnitudeTolerance);
    }

  const unsigned int spaceDimension = m_CostFunction->GetNumberOfParameters();

  m_Gradient = DerivativeType(spaceDimension);
  m_PreviousGradient = DerivativeType(spaceDimension);
  m_Gradient.Fill(0.0f);
  m_PreviousGradient.Fill(0.0f);

  this->SetCurrentPosition( GetInitialPosition() );
  this->ResumeOptimization();
}

/** Resume the optimization **/
void
RegularStepGradientDescentBaseOptimizer
::ResumeOptimization(void)
{
  m_Stop = false;

  this->InvokeEvent( StartEvent() );

  while ( !m_Stop )
    {
    if ( m_CurrentIteration >= m_NumberOfIterations )
      {
      m_StopCondition = MaximumNumberOfIterations;
```

```
      m_StopConditionDescription << "Maximum number of iterations ("
                                 << m_NumberOfIterations
                                 << ") exceeded.";
      this->StopOptimization();
      break;
      }

    m_PreviousGradient = m_Gradient;

    try
      {
      m_CostFunction->GetValueAndDerivative(
        this->GetCurrentPosition(), m_Value, m_Gradient);
      }
    catch ( ExceptionObject & excp )
      {
      m_StopCondition = CostFunctionError;
      m_StopConditionDescription << "Cost function error after "
                                 << m_CurrentIteration
                                 << " iterations. "
                                 << excp.GetDescription();
      this->StopOptimization();
      throw excp;
      }

    if ( m_Stop )
      {
      break;
      }

    this->AdvanceOneStep();

    m_CurrentIteration++;
    }
}

/** Stop optimization **/
void
RegularStepGradientDescentBaseOptimizer
::StopOptimization(void)
{
  m_Stop = true;
  this->InvokeEvent( EndEvent() );
}

/** Advance one Step following the gradient direction **/
```

```
void
RegularStepGradientDescentBaseOptimizer
::AdvanceOneStep(void)
{
  const unsigned int spaceDimension = m_CostFunction->GetNumberOfParameters();

  DerivativeType transformedGradient(spaceDimension);
  DerivativeType previousTransformedGradient(spaceDimension);
  ScalesType     scales = this->GetScales();

  if ( m_RelaxationFactor < 0.0 )
    {
    itkExceptionMacro(<< "Relaxation factor must be positive. Current value is "
       << m_RelaxationFactor);
    return;
    }

  if ( m_RelaxationFactor >= 1.0 )
    {
    itkExceptionMacro(<< "Relaxation factor must less than 1.0. Current value is "
       << m_RelaxationFactor);
    return;
    }

  // Make sure the scales have been set properly
  if ( scales.size() != spaceDimension )
    {
    itkExceptionMacro(<< "The size of Scales is "
                      << scales.size()
                      << ", but the NumberOfParameters for the CostFunction is "
                      << spaceDimension
                      << ".");
    }

  for ( unsigned int i = 0; i < spaceDimension; i++ )
    {
    transformedGradient[i]  = m_Gradient[i] / scales[i];
    previousTransformedGradient[i] =
      m_PreviousGradient[i] / scales[i];
    }

  double magnitudeSquare = 0;
  for ( unsigned int dim = 0; dim < spaceDimension; dim++ )
    {
    const double weighted = transformedGradient[dim];
    magnitudeSquare += weighted * weighted;
```

```
    }

  const double gradientMagnitude = vcl_sqrt(magnitudeSquare);

  if ( gradientMagnitude < m_GradientMagnitudeTolerance )
    {
    m_StopCondition = GradientMagnitudeTolerance;
    m_StopConditionDescription << "Gradient magnitude tolerance met after "
                               << m_CurrentIteration
                               << " iterations. Gradient magnitude ("
                               << gradientMagnitude
                               << ") is less than gradient magnitude tolerance ("
                               << m_GradientMagnitudeTolerance
                               << ").";
    this->StopOptimization();
    return;
    }

  double scalarProduct = 0;

  for ( unsigned int i = 0; i < spaceDimension; i++ )
    {
    const double weight1 = transformedGradient[i];
    const double weight2 = previousTransformedGradient[i];
    scalarProduct += weight1 * weight2;
    }

  // If there is a direction change
  if ( scalarProduct < 0 )
    {
    m_CurrentStepLength *= m_RelaxationFactor;
    }

  if ( m_CurrentStepLength < m_MinimumStepLength )
    {
    m_StopCondition = StepTooSmall;
    m_StopConditionDescription << "Step too small after "
                               << m_CurrentIteration
                               << " iterations. Current step ("
                               << m_CurrentStepLength
                               << ") is less than minimum step ("
                               << m_MinimumStepLength
                               << ").";
    this->StopOptimization();
    return;
    }
```

```
  double direction;
  if ( this->m_Maximize )
    {
    direction = 1.0;
    }
  else
    {
    direction = -1.0;
    }

  const double factor =
    direction * m_CurrentStepLength / gradientMagnitude;

  // This method StepAlongGradient() will
  // be overloaded in non-vector spaces
  this->StepAlongGradient(factor, transformedGradient);

  this->InvokeEvent( IterationEvent() );
}

const std::string
RegularStepGradientDescentBaseOptimizer
::GetStopConditionDescription() const
{
  return m_StopConditionDescription.str();
}

} // end namespace itk
```

A.2 Méthode de descente du gradient avec un pas régulier (Suite)

```
#ifndef _itkRegularStepGradientDescentOptimizer_hxx
#define _itkRegularStepGradientDescentOptimizer_hxx

#include "itkRegularStepGradientDescentOptimizer.h"

{
/** Advance one Step following the gradient direction  **/
/** This method will be overrided in non-vector spaces **/
void
RegularStepGradientDescentOptimizer
::StepAlongGradient(double factor,
                    const DerivativeType & transformedGradient)
```

```
{
  itkDebugMacro(<< "factor = " << factor << "  transformedGradient=
   " << transformedGradient);

  const unsigned int spaceDimension =
    m_CostFunction->GetNumberOfParameters();

  ParametersType newPosition(spaceDimension);
  ParametersType currentPosition = this->GetCurrentPosition();

  for ( unsigned int j = 0; j < spaceDimension; j++ )
    {
    newPosition[j] = currentPosition[j] + transformedGradient[j] * factor;
    }

  itkDebugMacro(<< "new position = " << newPosition);

  this->SetCurrentPosition(newPosition);
}
} // end namespace itk
```

A.3 Méthode de descente du gradient

```
#ifndef _itkGradientDescentOptimizer_hxx
#define _itkGradientDescentOptimizer_hxx

#include "itkGradientDescentOptimizer.h"

namespace itk
{
/** Constructor **/
GradientDescentOptimizer
::GradientDescentOptimizer()
{
  itkDebugMacro("Constructor");

  m_LearningRate = 1.0;
  m_NumberOfIterations = 100;
  m_CurrentIteration = 0;
  m_Maximize = false;
  m_Value = 0.0;
  m_StopCondition = MaximumNumberOfIterations;
  m_StopConditionDescription << this->GetNameOfClass() << ": ";
}

const std::string
```

```
GradientDescentOptimizer
::GetStopConditionDescription() const
{
  return m_StopConditionDescription.str();
}

void
GradientDescentOptimizer
::PrintSelf(std::ostream & os, Indent indent) const
{
  Superclass::PrintSelf(os, indent);

  os << indent << "LearningRate: "
     << m_LearningRate << std::endl;
  os << indent << "NunberOfIterations: "
     << m_NumberOfIterations << std::endl;
  os << indent << "Maximize: "
     << m_Maximize << std::endl;
  os << indent << "CurrentIteration: "
     << m_CurrentIteration;
  os << indent << "Value: "
     << m_Value;
  if ( m_CostFunction )
    {
    os << indent << "CostFunction: "
       << m_CostFunction;
    }
  os << indent << "StopCondition: "
     << m_StopCondition;
  os << std::endl;
  os << indent << "Gradient: "
     << m_Gradient;
  os << std::endl;
}

/** Start the optimization **/
void
GradientDescentOptimizer
::StartOptimization(void)
{
  itkDebugMacro("StartOptimization");

  m_CurrentIteration   = 0;

  this->SetCurrentPosition( this->GetInitialPosition() );
  this->ResumeOptimization();
```

```
}

/** Resume the optimization **/
void
GradientDescentOptimizer
::ResumeOptimization(void)
{
  itkDebugMacro("ResumeOptimization");

  m_Stop = false;

  m_StopConditionDescription.str("");
  m_StopConditionDescription << this->GetNameOfClass() << ": ";
  InvokeEvent( StartEvent() );
  while ( !m_Stop )
    {
    try
      {
      m_CostFunction->GetValueAndDerivative(
        this->GetCurrentPosition(), m_Value, m_Gradient);
      }
    catch ( ExceptionObject & err )
      {
      // An exception has occurred.
      // Terminate immediately.
      m_StopCondition = MetricError;
      m_StopConditionDescription << "Metric error";
      StopOptimization();

      // Pass exception to caller
      throw err;
      }

    if ( m_Stop )
      {
      m_StopConditionDescription << "StopOptimization() called";
      break;
      }

    AdvanceOneStep();

    m_CurrentIteration++;

    if ( m_CurrentIteration >= m_NumberOfIterations )
      {
      m_StopConditionDescription << "Maximum number of iterations ("
```

```
                                << m_NumberOfIterations
                                << ") exceeded.";
      m_StopCondition = MaximumNumberOfIterations;
      StopOptimization();
      break;
      }
    }
}

/** Stop optimization **/
void
GradientDescentOptimizer
::StopOptimization(void)
{
  itkDebugMacro("StopOptimization");

  m_Stop = true;
  InvokeEvent( EndEvent() );
}

/** Advance one Step following the gradient direction **/
void
GradientDescentOptimizer
::AdvanceOneStep(void)
{
  itkDebugMacro("AdvanceOneStep");

  double direction;
  if ( this->m_Maximize )
    {
    direction = 1.0;
    }
  else
    {
    direction = -1.0;
    }

  const unsigned int spaceDimension =  m_CostFunction->GetNumberOfParameters();

  const ParametersType & currentPosition = this->GetCurrentPosition();

  ScalesType scales = this->GetScales();

  // Make sure the scales have been set properly
  if ( scales.size() != spaceDimension )
    {
```

```
    itkExceptionMacro(<< "The size of Scales is "
                      << scales.size()
                      << ", but the NumberOfParameters for the CostFunction is "
                      << spaceDimension
                      << ".");
    }

  DerivativeType transformedGradient(spaceDimension);

  for ( unsigned int j = 0; j < spaceDimension; j++ )
    {
    transformedGradient[j] = m_Gradient[j] / scales[j];
    }

  ParametersType newPosition(spaceDimension);
  for ( unsigned int j = 0; j < spaceDimension; j++ )
    {
    newPosition[j] = currentPosition[j]
                     + direction * m_LearningRate * transformedGradient[j];
    }

  this->SetCurrentPosition(newPosition);

  this->InvokeEvent( IterationEvent() );
}
} // end namespace itk
```